Savitribai Phule Pune University
(Formerly University of Pune)

S.Y.B.Sc.(Computer Science)

With

Major: Computer Science

(Faculty of Science and Technology)

(For Colleges Affiliated to Savitribai Phule Pune University)

Choice Based Credit System (CBCS) Syllabus Under National Education Policy (NEP)

To be implemented from Academic Year 2025-2026

Title of the Course: B.Sc.(Computer Science)

Savitribai Phule Pune University
(Formerly University of Pune)

S.Y.B.Sc.(Computer Science) Semester IV

WORKBOOK

Course Code: CS-253-MJ-P

(Lab Course based on CS-251-MJ-T & CS-252-MJ-T)

WORKBOOK

S.Y.B.Sc. (Computer Science)

Data Structures II
and
Database Management Systems II

Course Type: Major Core Course Code: CS-253-MJ-P
Course Title: Lab Course based on CS-251-MJ-T & CS-252-MJ-T

SEMESTER IV

Student Name:			
College:			
Roll No:		**Exam Seat No:**	
Year:		**Division:**	

BOARD OF STUDIES

1. Dr. Patil Ranjeet	2. Dr. Joshi vinayak
3. Dr. Wani Vilas	4. Dr. Mulay Prashant
5. Dr. Sardesai Anjali	6. Dr. Shelar Madhukar
7. Dr. Bharambe Manisha	8. Dr. Deshpande Madhuri
9. Dr. Kardile Vilas	10. Dr. Korpal Ritambhara
11. Dr. Gangurde Rajendra	12. Dr. Manza Ramesh
13. Dr. Bachav Archana	14. Dr. Bhat Shridhar

Co-ordinators

- **Dr. Prashant Mulay**

Annasaheb Magar College, Hadapsar , Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

- **Dr. Sardesai Anjali**

Modern College of Arts, Science and Commerce , Shivajinagar, Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

Editor

Dr. Prashant Mulay, Annasaheb Magar College, Hadapsar , Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

Prepared by:

Ms. Sunita J. Saykar	Annasaheb Magar College, Hadapsar, Pune.
Ms. Vinita B. Kadlag	Annasaheb Magar College, Hadapsar, Pune.

Table of Contents for CS-203-MJ-P

Introduction

About the workbook

This workbook is intended to be used by S.Y.B.Sc (Computer Science) students for the Lab Course based on CS-251-MJ-T & CS-252-MJ-T Data structures II and Database Management Systems II.

Workbook is divided in two sections.

1. First section contains assignments on Data Structures II
2. Second section contains assignments on Database Management Systems II.

Data structures is core subject of computer science curriculum and hands-on laboratory experience is critical to the understanding of theoretical concepts studied as part of this course. Study of any programming language is incomplete without hands on experience of implementing solutions using programming paradigms and verifying them in the lab. This workbook provides rich set of problems covering the basic algorithms as well as numerous computing problems demonstrating the applicability and importance of various data structures and related algorithms.

The objectives of this book are

- Defining the scope of the course
- Bringing uniformity in the way the course is conducted across different colleges
- Continuous assessment of the course
- Bring variation and variety in experiments carried out by different students in a batch
- Providing ready reference for students while working in the lab
- Catering to the need of slow paced as well as fast paced learners

How to use this workbook?

The **Data structures-II** practical syllabus is divided into five assignments. Each assignment has problems divided into three sets A, B and C.

- Set A is used for implementing the basic algorithms or implementing data structure along with its basic operations. **Set A is mandatory.**
- Set B is used to demonstrate small variations on the implementations carried out in set A to improve its applicability. Depending on the time availability the students should be encouraged to complete set B.
- In Set C, Students should spend additional time either at home or in the Lab and solve these problems so that they get a deeper understanding of the subject.

The **DBMS-II** practical syllabus is divided into five assignments. For assignment number 2 to assignment number 5 includes five sets A, B, C, D and E. Students should solve any **three Sets** of them compulsory.

Instructions to the students

Please read the following instructions carefully and follow them.

- Students are expected to carry workbook during every practical.
- Students should prepare oneself beforehand for the Assignment by reading the relevant material.

- Instructor will specify which problems to solve in the lab during the allotted slot & student should complete them and get verified by the instructor. However, student should spend additional hours in Lab and at home to cover as many problems as possible given in this work book.

- Students will be assessed for each exercise on a scale from 0 to 5

➢	Not done	0
➢	Incomplete	1
➢	Late Complete	2
➢	Needs improvement	3
➢	Complete	4
➢	Well Done	5

Instruction to the Practical In-Charge

- Explain the assignment and related concepts.
- Choose appropriate problems to be solved by students. Set A is mandatory. Choose problems from set B depending on time availability. Discuss set C with students and encourage them to solve the problems by spending additional time in lab or at home.
- Make sure that students follow the instruction as given above.
- You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- The value should also be entered on assignment completion page of the respective Lab course.

Instructions to the Lab administrator and Exam guidelines

- You have to ensure appropriate hardware and software is made available to each student.
- Do not provide Internet facility in Computer Lab while examination
- Do not provide pen drive facility in Computer Lab while examination.

The operating system and software requirements are as given below:

- Operating system: Linux
- Editor: Any Linux based editor like vi, gedit etc.
- Compiler: cc or gcc
- PostgreSQL

Assignment Completion Sheet

Section I Data Structures I

Sr. No	Assignment Name	Marks (Out of 5)	Signature
1	**Binary Search Tree and Traversals**		
2	**Binary Search Tree Operations and Applications**		
3	**Graph implementation**		
4	**Graph Applications**		
5	**Hash Table**		
a. Total out of 30			

Section II Database Management System I

Sr. No.	Assignment Name	Marks (out of 5)	Signature
1.	**Stored Procedure**		
2.	**Function**		
3.	**Cursors**		
4.	**Exception Handling**		
5.	**Trigger**		
b. Total out of 30			
a+b (Marks out of 60 to be converted to out of 15)			

This is to certify that **Mr/Ms** _____

University Exam Seat Number _____have successfully and satisfactorily completed and submitted the course work for **S.Y.B.Sc.(CS)** Lab course CS-203-MJ-P Semester III prescribed by Savitribai Phule Pune University during the academic year _____.

Instructor **Head of Department**

Internal Examiner **External Examiner**

Section I

Data Structures-II

Assignment 1: Binary Search Tree and Traversals

Definition:

A binary search tree is a binary tree, which is either empty or in which each node contains a key that satisfies following conditions:

1) For every node X, in the tree the values of all the keys in its left subtree are smaller than the key value in X.

2) For every node X, in the tree the values of all the keys in its right subtree are larger than the key value in X.

Representation of Binary Tree:

1. Static Representation of Binary Tree:

One of the way to represent binary tree using array is to store nodes level by level, starting from the zero level where the root is present. Such representation needs sequential numbering of the nodes starting with nodes on level zero, then those on level 1 and so on. Already, we have defined complete tree. A complete binary tree of height h has $(2^{h+1} - 1)$ nodes in it. The nodes can be stored in one dimensional array, TREE, with the node numbered at location TREE(i). A array of size $(2^{h+1} - 1)$ or $2^d - 1$ (where d = no. of levels) is required.

The root node is stored in the first memory location as the first element in the array. Following rules can be used to decide the location of any i^{th} node of a tree:

For any node with index i, $1 \le i \le n$;

a) PARENT (i) = $\left\lfloor \frac{i}{2} \right\rfloor$ if $i \neq 1$

If i = 1 then it is root which has no parent

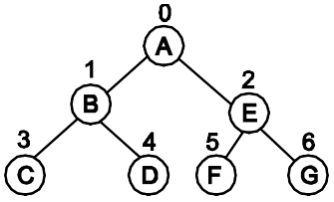
b) LCHILD (i) = 2 * i, if $2i \le n$

If 2i > n, then i has no left child

c) RCHILD(i) = 2i + 1, if $2i + 1 \le n$

If (2i + 1) > n, then i has no right child

Example: Consider the given Binary tree:

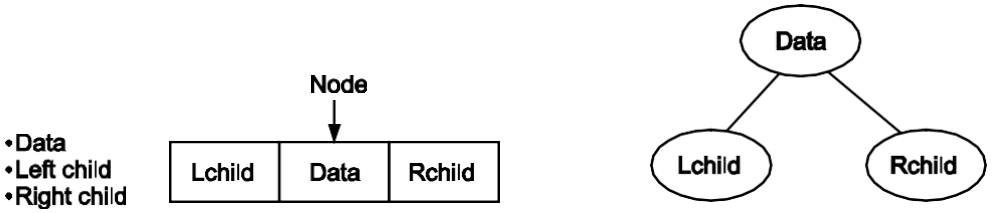


The representation of the above binary tree using array is as followed:

1	2	3	4	5	6	7
A	B	C	D	E	F	G

2. Linked Representation of Binary Tree:

Another way to represent a binary tree is linked list, which is more memory efficient than the array representation. All nodes should be allocated dynamically. Each node with data and link fields. The root pointer points to the tree in memory.
Each node consists of three fields, Lchild, Data and Rchild.



Linked representation Binary tree

Binary Search Tree (BST):

It is a binary tree with the property that the value in a node is greater than any value in a node's left subtree and less than any value in the node's right subtree. The Operations on Binary Search Tree are:

init (T)	creates an empty Binary search tree by initializing T to NULL
insert (T, x)	inserts the value x in the proper position in the Binary search tree
search (T, x)	searches if the value x is present in the search tree
inorder (T)	displays the node using inorder traversal of binary search tree
postorder (T)	displays the node using postorder traversal of binary search tree
preorder (T)	displays the node using preorder traversal of binary search tree
delete(T, x)	Deletes node x from binary search tree

Consider the structure of a node of a BST*:*

```
typedef struct BST
{
        struct BST * Lchild;
        int data;
        struct BST * Rchild;
}BSTnode;
```

Insert (T,x): Consider T is root node and x is element to be inserted in BST

Algorithm Steps:

1. t = root, flag = false
2. while (t ≠ null) & (flag = false) do
   Case 1: key < t -> data
   t1 = t;
   t = t ->Lchild;

Case 2: key > t -> data
t1 = t
t = t ->Rchild
Case 3: t -> data = x
flag = true
display "item already exist"
break
end case
End while

3. If (t = null) then
new = getnode (node) //create node
new -> data = x
new ->Lchild = null //initialize a node
new ->Rchild = null
if (t1 -> data <x) then //insert at right
t1 ->Rchild = new
else t1 ->Lchild = new //insert at left
endif
4. stop

Search(T,x): Consider T is a root node and x is an element to be searched in BST.

Algorithm steps:

1. Initialize t = root, flag = false
2. while (t ≠ null) and (flag = false) do
Case 1: t->data = key
flag = true //successful search
Case 2: Key < t -> data
t = t ->Lchild // goto left subtree
Case 3: Key > t -> data
t = t ->Rchild //goto right subtree
End case
End while
3. If (flag = true) then
display "Key is found at node", t
else
display "Key is not exist"
end if;
4. Stop

Traversal Methods (Recursive Algorithm):

Consider T is root node of BST which is passed as argument to a function when function is called from main() function.

inorder(T):

Algorithm steps:

1. If T is not empty then
inorder (T->Lchild)

display T->data
inorder(T->Rchild)

2. stop

preorder(T):

Algorithm steps:

1. If T is not empty then
   display T->data
   preorder (T->Lchild)
   preorder(T->Rchild)
2. stop

postorder(T):

Algorithm steps:

1. If T is not empty then
   postorder (T->Lchild)
   postorder(T->Rchild)
   display T->data
2. stop

Delete (T,x): Consider T is root node and x is an element to be deleted from BST**.** There are three cases with respective to node to be deleted:

1. x is a leaf node.
2. x has one child.
3. x has both child nodes.

Algorithm Steps:

1. t = root, flag = false
2. while (t ≠ null) and (flag = false) do
   Case 1: key < t->data
   parent = t
   t = t ->Lchild
   Case 2: key > t -> data
   parent = t
   t = t ->Rchild
   Case 3: t -> data = key
   flag = true
   end case
   end while
3. If flag = false
   Then display "item not exist".
   exit.
   /* case 1 if node has no child */
4. If (t ->Lchild= null) and (t ->Rchild= null) then
   if (parent ->Lchild= t) then
   parent ->Lchild = null //set pointer to its parent when node is left child
   else
   parent ->Rchild = null

5. /* case 2 if node contains one child */
   If (parent ->Lchild = t) then //when node contains only one child
      if (t ->Lchild = null) then //if node is left child
         parent ->Lchild = t ->Rchild
      else
         parent ->Lchild= t ->Lchild
      endif
   else
      if (parent ->Rchild = t) then
         if (t ->Lchild = null) then
            parent ->Rchild = t ->Rchild
         else
            parent ->Rchild = t ->Lchild
         endif
      endif
   endif
6. /* Case 3 if node contains both child */
   t1 = succ(t) //find inorder successor of the node
   key1 = t1 -> data
   Delete(key1) //delete inorder successor
   t -> data = key //replace data with the data of an order successor
7. Stop.

Other operations on a binary search tree include counting leaf and non leaf nodes in the tree.
count (T)(Recursive Definition): This will give total number of nodes of BST

Algorithm Steps:

1. counter=0
2. If (T≠NULL) then
   counter=counter+1
   count (T->Lchild)
   count(T->Rchild)
   2. stop

Similarly, conditions for checking leaf node and non-leaf can be written and respective counters can be incremented.

Set A

a) Implement a Binary search tree (BST) library (btree.h) with operations – create, search, insert, inorder, preorder and postorder. Write a menu driven program that performs the above operations.

b) Write a program which uses binary search tree library and counts the total nodes and total leaf nodes in the tree.
int count(T) – returns the total number of nodes from BST
int countLeaf(T) – returns the total number of leaf nodes from BST

Set B

a) Write a C program which uses Binary search tree library and implements following function with recursion:
int copy(T) – create another BST which is exact copy of BST which is passed as parameter.
int compare(T1, T2) – compares two binary search trees and returns 1 if they are equal and 0 otherwise.

Set C

a) Write a C program which uses Binary search tree library and implements following two functions:
int sumodd(T) – returns sum of all odd numbers from BST
int sumeven(T) – returns sum of all even numbers from BST
mirror(T) – converts given tree into its mirror image.

b) Write a function to delete an element from BST.

c) What modifications are required in search function to count the number of comparisons required?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Date:

Assignment 2: Binary Search Tree Operations and Applications

BST Operations

1. Insertion

- Start at root
- If key < current node → go left
- If key > current node → go right
- Insert when the correct NULL position is found

2. Searching

- Compare key with current node
- If equal → found
- If smaller → search left
- If larger → search right

3. Deletion

A node has **3 cases**:

Case 1: Leaf Node

- Simply remove the node

Case 2: Node with One Child

- Replace node with its child

Case 3: Node with Two Children

4. Traversal

Inorder (LNR)

Produces **sorted output**

Left → Node → Right

Preorder (NLR)

Node → Left → Right

Used to create a copy of the tree

Postorder (LRN)

Left → Right → Node

Used to delete/free the tree

The level order traversal **levelwisedisplay (T)** traverses the tree levelwise starting from the root.

Here queue data structure is required to store node address. The variables total represents total number of nodes of BST, variable level represents number of level in BST and variable count represents total number of node in each level.

Algorithm Steps:

1. temp = root
2. add(Q, temp) //insert temp in queue
3. add(Q, NULL)//insert NULL in queue
4. if queue is not empty then
   temp = delete(Q)
   else goto 6
5. if (temp≠NULL) then
   total=total+1
   cnt=cnt+1
   display t -> data
   if (temp ->Lchild≠NULL) then

```
                add(Q, T->Lchild)
        if (T->Rchild≠NULL) then
                add(Q, T->Rchild)
                goto step 4
    else
        if queue is not empty then
                display "\n" ( newline character )
            display cnt
         cnt = 0
            level=level+1
add(Q,NULL)
      else goto step4
```

6. Stop

The Binary tree is widely used in a variety of applications. It is used in a sorting method called Heap Sort. This sorting method uses a special type of binary tree called "Heap". A max-heap is a full or complete binary tree such that each parent has a value greater than its children. A set of n elements can be considered as a binary tree implemented using array. This tree is converted into a Heap and then sorted.

Algorithm Heapsort

Step 1.Start
Step 2. Accept n elements in array A.
Step 3. Convert array A into a heap
Step 4. last = n – 1, top = 0.
Step 5. Interchange A[top] and A[last]
Step 6. last = last – 1
Step 7. Heapify (A, top, last), i.e. recreate heap
Step 8. If last > 0
goto Step 5
Step 9. Stop

Algorithm Heapify (A, top, last)

1. Start
2. key = A[top]
3. j = 2∗top +1 i.e. j is at the left child
4. If A[j] < A[j+1] i.e. j points to the larger child
   j = j + 1
5. If key < A[j] i.e. if parent < child
   Interchange A[top] and A[j]
   Heapify (A,j,n)
6. Stop

Set A

a) Write a C program which uses Binary search tree library and display the following
   i. Node value at each level
   ii. Count of node at each level
   iii. Total levels in the tree.

b) Write a C program to find the minimum and maximum values in a Binary Search Tree

Set B

a) Write a program to sort n randomly generated elements using Heapsort method.

b) Write a C program to maintain a phonebook using Binary Search Tree by name where each node contain contact name and phone number.

c) Write a C program to find the height of the tree and check whether given tree is balanced or not.

Set C

a) Which data structure is require to display nodes of Binary Search Tree depth wise?
b) Write a C program to displays nodes of Binary Search Tree depth wise.
c) Write a C program to compare two binary search trees (node wise comparison).
d) How to implement mirror() and copy() functions without recursion?
e) How to convert singly linked list to binary search tree?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge
Date:

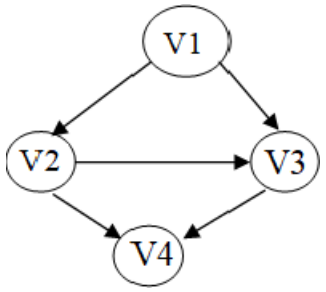
Assignment 3: Graph Implementations

1] Graph as an Adjacency Matrix:-

A graph consists of a set of vertices and a set of edges. We can write a graph as G=(V,E), where V is a set of nodes (vertices) and E is a set of edges (arcs).

The one way of representing graphs is adjacency matrix representation.

In adjacency matrix representation of a Graph with n vertices and e edges, a two dimensional n x n array ,say a , is used , with the property that a[i,j] equals 1 if there is an edge from i to j and a[i,j] equals 0 if there is no edge from i to j.

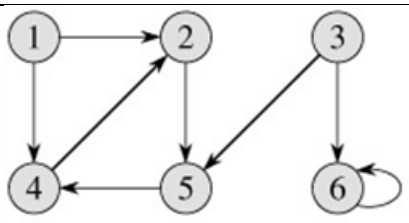
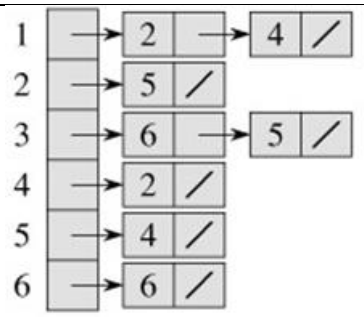
Graph	Adjacency Matrix
	A44= V1 V2 V3 V4 V1: 0 1 1 0 V2: 0 0 1 1 V3: 0 0 0 1 V4: 0 0 0 0

The usual operations on graph are:

Indegree(i) – returns the indegree (the number of edges ending on) of the ith vertex Outdegree(i) – returns the outdegree(the number of edges moving out) of the ith vertex) displayAdjMatrix – displays the adjacency matrix for the graph

2] Graph as an Adjacency List:-

Another way of representing graphs is adjacency list representation. In adjacency list representation of a graph with n vertices and e edges, there are n linked lists, one list for each vertex in the graph.

Graph	Adjacency List
	

To implement graph as an adjacency list, we create an array of lists. The usual operations on graph are:

Indegree(i) – returns the indegree (the number of edges ending on) of the ith vertex Outdegree(i) – returns the outdegree(the number of edges moving out) of the ith vertex) display_adj_List() – displays the adjacency list for the graph

Set A

a) Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix.

b) Write a C program that accepts the vertices and edges of a graph. Create and display adjacency list also print indegree, outdegree and total degree of all vertex of graph.

Set B

a) Write a C program that accepts the vertices and edges of a graph and store it as an **adjacency matrix**. Implement function to traverse the graph using Breadth First Search (BFS) and Depth First Search (DFS) traversal.

b) Write a C program that accepts the vertices and edges of a graph and store it as an **adjacency list.** Implement function to traverse the graph using Breadth First Search (BFS) and Depth First Search (DFS) traversal.

Set C

a) Which data structure is used to implement Depth First Search?
b) Where the new node is appended in Breadth-first search of OPEN list?
c) What is complete graph?
d) Which data structure is used to implement adjacency list method?
e) Compare adjacency matrix and adjacency list.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Date:

Assignment 4: Graph Applications

Topological Sorting:

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that if there is an edge directed towards vertex v_j from vertex v_i, then v_i comes before v_j. Topological sorting for a graph is not possible if the graph is not a DAG. There can be more than one topological sorting for a graph.

For topological sort to perform we need to find adjacency matrix. Example: The **topological sort** algorithm takes a directed graph and returns an array of the nodes where each node appears before all the nodes it points to.

Algorithm

1. Begin
2. Create a graph
3. Find in degree of each vertex
4. Insert vertices of in degree 0 in queue Q
5. While queue is not empty and all vertices are not covered
   - Add vertex v to sorted array Topo_sort
   - Delete all outgoing edges from vertex v
6. If all vertices are covered then display Topo_sort array
   Else display message "Graph contains cycle, no topological ordering is possible".
7. End

Minimum Spanning Tree

A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. An example is a cable company wanting to lay line to multiple neighborhoods; by minimizing the amount of cable laid, the cable company will save money.

A **tree** has one path joins any two vertices. A **spanning tree** of a graph is a tree that:

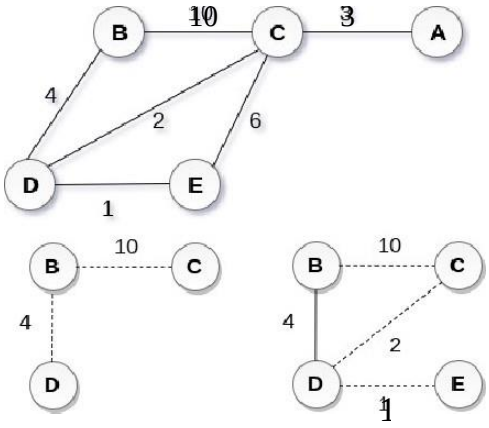
- Contains all the original graph's vertices.
- Reaches out to (spans) all vertices.
- Is acyclic means the graph doesn't have any nodes which loop back to itself.

The minimum spanning tree from a graph is found using the following algorithms:

A. Prim's Algorithm
B. Kruskal's Algorithm

A. Prims *Algorithm*

Example:



Step 1 step 2 step 3

Step 4 **step 5**

Algorithm:

1) Start
2) Create a set *mstSet* that keeps track of vertices already included in MST.
3) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign
key value as 0 for the first vertex so that it is picked first.
4) While mstSet doesn't include all vertices
a) Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
b) Include *u* to mstSet.
c) Update key value of all adjacent vertices of *u.*

To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v*
5) End

B. Kruskal's Algorithm

1. Create a priority queue Q that contains all the edges of the graph.
2. Repeat Steps 4 and 5 while Q is NOT EMPTY
3. Remove an edge from Q

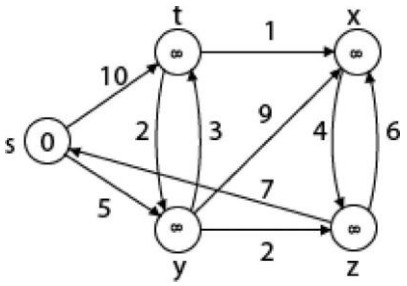
4. IF the edge obtained in Step 4 connects two different trees, then Add it to the forest (for combining two trees into one tree).

   ELSE
   Discard the edge
5. END

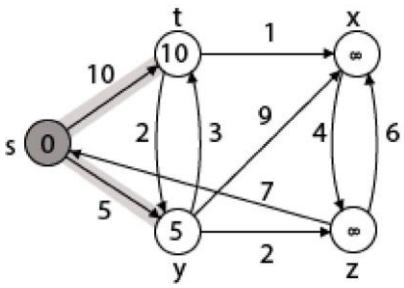
Dijkstra's algorithm

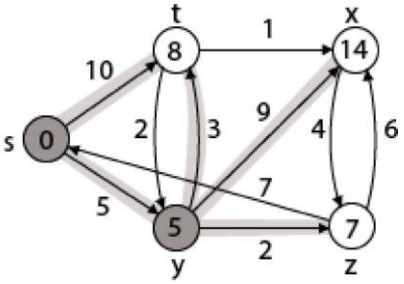
Dijkstra algorithm is a greedy algorithm. It finds a shortest path tree for a weighted undirected graph. Dijkstra's algorithm finds the solution for the single source shortest path problems only when all the edge-weights are non-negative.

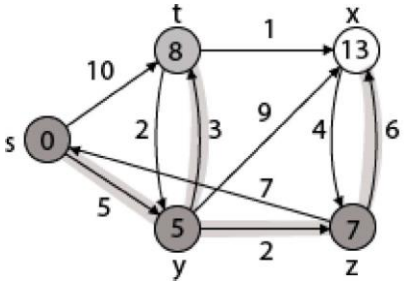
Example:

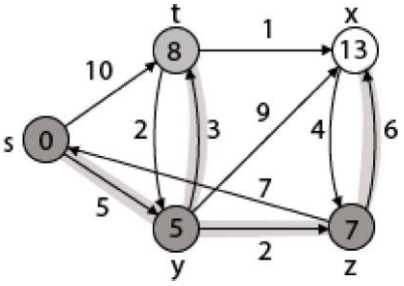


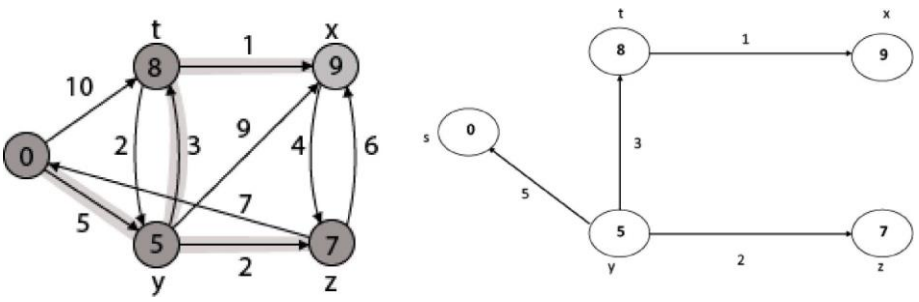
All vertices are scanned one by one to find out adjacent vertices. Calculate the distance of each adjacent to the source vertices. A stack is maintained, which contains those vertices which are selected after computation of shortest distance. Now find the adjacent of s that are t and y. Find shortest distances of t and y. It is found that y is with shortest distance.











A stack is maintained, which contains those vertices which are selected after computation of shortest distance. Now find the adjacent of s that are t and y. Find shortest distances of t and y. It is found that y is with shortest distance.

1. Create a set shortPath to store vertices that come in the way of the shortest path tree.
2. Initialize all distance values as INFINITE and assign distance values as 0 for source vertex so that it is picked first.
3. Loop until all vertices of the graph are in the shortPath.
   3.1 : Take a new vertex that is not visited and is nearest.
   3.2 : Add this vertex to shortPath.
   3.3 : For all adjacent vertices of this vertex update distances. Now check every adjacent of V, if sum of distance of u and weight of edge is else the update it.

Floyd-Warshall Algorithm:

This algorithm is used for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).This algorithm follows the dynamic programming approach to find the shortest paths.

The solution matrix is initialized same as the input graph matrix. Then update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When vertex k is selected as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices. For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

- k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
- k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j].

Input − The cost matrix of given Graph.

Output -- Matrix to for shortest path between any vertices to any vertex.

```
Begin
  for k := 0 to n, do
    fori := 0 to n, do
      for j := 0 to n, do
        if cost[i,k] + cost[k,j] < cost[i,j], then
          cost[i,j] := cost[i,k] + cost[k,j]
      done
    done
  done
  display the current cost matrix
End
```

Set A

a) Write a C program for the implementation of Topological sorting.

b) Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.

c)Write a C program for the Implementation of Kruskal's Minimum spanning tree algorithm.

Set B

a) Write a C program for the implementation of Dijkstra's shortest path algorithm for finding shortest path from a given source vertex using adjacency cost matrix.

b) Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix.

Set C

a) A graph may not have an edge from a vertex back to itself(self edges or self loops). Given an adjacency matrix representation of a graph, how to know if there are self edges?

b) Differences between Prim's and Kruskal's algorithms.

c) Give two real-world applications of **shortest-path algorithms.**

d) In what situations is a **minimum spanning tree** more useful than shortest paths?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Date:

Assignment 5: Hash Table

A hash table or hash map is a data structure that efficiently stores and retrieves data from memory. Hash table is a data structure that represents data in the form of key-value pairs. Each key is mapped to a value in the hash table. The keys are used for indexing the values/data. A similar approach is applied by an associative array. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found. Ideally, the hash function will assign each key to a unique bucket, but most hash table designs employ an imperfect hash function, which might cause hash collisions where the hash function generates the same index for more than one key.

Operations to be performed

a. Initialize the Hash Bucket
b. Insert elements in the hash table
c. Search elements from the hash table
d. Delete an element from the hash table
e. Display the contents of hash table

Algorithm

1. Create hash table (array of max size)
2. Take a value to be stored in hash table as input.
3. Apply appropriate has function to generate a key(index) based on the value
4. Using the generated index, access the data located in that array index.
5. In case of absence of data, insert the data item (value) into it and increment the size of hash table.
6. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item.
   Note: This probing will continue until we reach the same element again (from where we began probing)
7. To display all the elements of hash table, element at each index is accessed (via for loop).
8. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements until we find key or an empty space where not a single data has been entered (means data does not exist in the hash table).
9. Exit

Algorithm | Insert data into the separate chain

1. Declare an array of a linked list with the hash table size.
2. Initialize an array of a linked list to NULL.
3. Find hash key.
4. If chain[key] == NULL
   Make chain[key] points to the key node.
5. Otherwise(collision),
   Insert the key node at the end of the chain[key].

Algorithm Searching a value from the hash table

1. Get the value
2. Compute the hash key.
3. Search the value in the entire chain. i.e. chain[key].
4. If found, print "Search Found"
5. Otherwise, print "Search Not Found"

Set A

a) Write a program to implement various types of hash functions which are used to place the data in a hash table

a. Division Method
b. Mid Square Method
c. Digit Folding Method

Accept n values from the user and display appropriate message in case of collision for each of the above functions.

b) Write a C program to implement a hash table using **separate chaining** with linked lists with following operation.

a. Insert a key
b. Search a key
c. Delete a key
d. Display the hash table

Set B

a) Write a C program to implement a hash table using **open addressing with linear probing**.perform the following operations.(Assume all keys are positive integers)

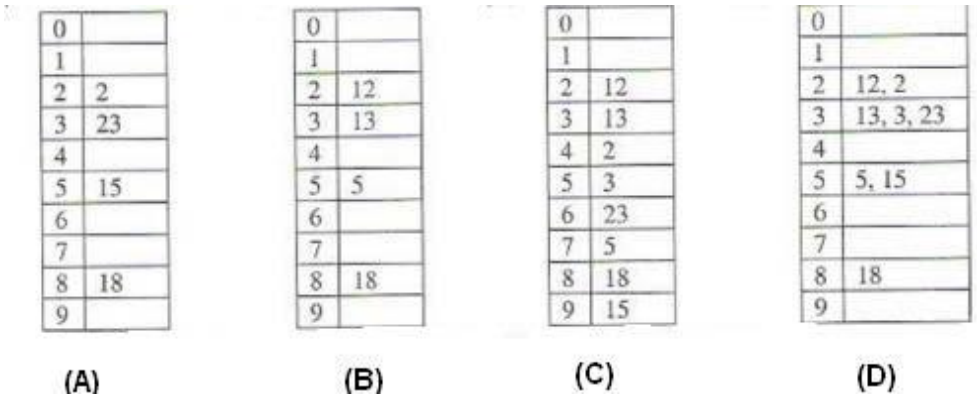
a. Insert
b. Search
c. Delete
d. Display

b) Write a C program to implement a hash table using **Open Addressing with Quadratic Probing**. Perform the following operations. (Assume all keys are positive integers)

a. Insert
b. Search
c. Delete
d. Display

Set C

a) Calculate the time complexity of hash table with Linear Probing
b) The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and linear probing. What is the resultant hash table?



0	
1	
2	2
3	23
4	
5	15
6	
7	
8	18
9	

(A)

0	
1	
2	12
3	13
4	
5	5
6	
7	
8	18
9	

(B)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(C)

0	
1	
2	12, 2
3	13, 3, 23
4	
5	5, 15
6	
7	
8	18
9	

(D)

Which data structure is appropriate for simple chaining? Why?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Date:

Section II

DBMS-II

Assignment 1: Stored Procedure

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.

A subprogram can be created at the schema level, Inside a package, Inside a PL/SQL block.

At the schema level, subprogram is a standalone subprogram. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A subprogram created inside a package is a packaged subprogram. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter 'PL/SQL - Packages'.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms −

- Functions − These subprograms return a single value; mainly used to compute and return a value.
- Stored Procedure − These subprograms do not return a value directly; mainly used to perform an action.

Each PL/SQL subprogram has a name, and may also have a parameter list.

Sr.No.	Parts & Description
1	**Declarative Part** It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.
2	**Executable Part** This is a mandatory part and contains statements that perform the designated action.
3	**Exception-handling** This is again an optional part. It contains the code that handles run-time errors.

```
CREATE [OR REPLACE] PROCEDURE procedure_name(parameter_list) LANGUAGE
language_name
AS $$
stored_procedure_body;
$$;
```

Create Procedure Syntax

Example 1:

```
CREATE OR REPLACE FUNCTION sum_n_product3(IN x int,IN y int, OUT sum int, OUT prod int) AS $$ BEGIN
IF x < 2 THEN
RAISE WARNING 'information message %', now(); RAISE
NOTICE 'information message %', now(); RAISE INFO
'information message %', now();
END IF;
sum := x + y; prod
:= x * y; END;
$$ LANGUAGE plpgsql
```

Solve the Following:

SET A

a. Write a procedure to display addition, subtraction and multiplication of three numbers.
b. Write a procedure to display division of two numbers use raise to display error messages.

c.Create table Department (dno, dname, empname, city).
i) Write a procedure to insert values in Department table.
ii) Write a procedure to display all employees working in ‘Pune’ city.

SET B

A) Consider the following relationship
Route(route_no, source, destination, no_of_station) Bus
(bus_no, capacity, depot_name)
Relationship between Route and Bus is **one-to-many**

a. Write a procedure which display all bus details for a given route.
b. Write a procedure to update source of route no 101

B) Consider the following relationship

Patient (p_no, p_name, p_addr) Doctor
(d_no, d_name, d_addr, city)
Relationship between Patient and Doctor is **many-to-many** with descriptive attribute disease and no_of_visits.

a. Write a procedure which display patient detail who has visited more than 3 times to the given doctor for 'Diabetes'.
b. Write a procedure which displays total number of visits of Dr.Kumar.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Assignment 2 - Stored Functions

Stored functions are user defined functions, that are created using the CREATE FUNCTION statement. The functions, thus created, are called stored functions because they are stored as database objects within the PostgreSQL database. The CREATE FUNCTION command names the new function, states its arguments and return type.

Creating a stored function:

CREATE FUNCTION identifier (arguments) RETURNS type AS'
`DECLARE
Variable–declarations; [……] BEGIN
Statement; [.....]
END ;
' LANGUAGE 'plpgsql';

Argument variables:

PL/pgSQL functions can accept argument variables of different types. Function arguments allow a user to pass information into a function, that the function may need.

Each function argument that is received by a function is incrementally assigned to an identifier that begins with the dollar ($) sign and is labeled with the argument number. Thus the identifier $1 is used for the first argument, $2 is used for the second argument and so on an so forth. PL/pgSQL allows us to create variable aliases. Aliases are created using the ALIAS keyword and it gives us the ability to provide an alternate variable to use when referencing argument variables. All aliases should be declared within the DECLARE section of a block before they are used.

Returning variables:

PL/pgSQL functions must return a value that matches the data type specified as the return type during the function definition. Values are returned from a function using the RETURN statement.

Attributes:

PL/pgSQL provides variable attributes that basically assists or helps the database programmer , when working with database objects. These attributes are %TYPE and %ROWTYPE.

The TYPE attribute : Used to declare a variable with the data type of a referenced database object (a table column).

Syntax :Variable_nametable_name.column_name%TYPE ;

The %ROWTYPEattribute : Used to declare a PL/pgSQL record variable to have the same structure as the rows in a table , that we specify in the function block.

Syntax :Variable_nametable_name%ROWTYPE;

Controlling Program Flow:

Most programming languages provides different ways of controlling the flow of execution, within a program. PL/pgSQL also provides various statements using which a programmer can control the way actions will be executed within a PL/pgSQL function code. PL/pgSQL provides

conditional statements and control loops for the same.

	Statement	**Syntax**

1	The IF/THEN statement	If < Condition> Then
		statement;
		[..]
		Else
		Statement; [..]
		End if;

	Nested If-else if	If Then
		Statement; [..]
		Else if Then
		Statement;
		[….]
		Else if Then
		Statement; [……]
		Else Statement;
		[….]
		End if;
		End if;
		End if;

	Loops	LOOP Statement; [……] EXIT [label] WHEN END LOOP;
	While Loop	While Condition Loop Statement; […….] End loop;
	For Loop	For Loop–Index In {Reverse} expression1 ..... expression2 Loop Statement; [..] End loop;
	For Loop (iterate through a query resultset)	For {record_variable | %rowtype_variable } IN select_statement LOOP Statement; [...] END LOOP

NOTE: Solve any three sets from the Following:

SET A

Using If.-Then-else,case,for,while and unconditional loops

1. Find minimum and maximum from two numbers
2. Check the number is positive, negative or zero.
3. Find maximum and minimum from three numbers
4. Find number is even or odd
5. Find sum of first 20 numbers (using unconditional loop)
6. Display all even numbers from 1 to 50.
7. Find sum and average of first n numbers using conditional loop(while)
8. Count odd numbers from given range (m to n) (for)
9. Search the given number is in given range.
10. Display a number in word (Using Case) and loop.

SET B

Bank database

Consider the following database maintained by a Bank. The Bank maintains information about its branches, customers and their loan applications.

Following are the tables:

branch (bid integer, brname char (30), brcity char (10))

customer (cno integer, cname char (20), caddr char (35), city char(20))

loan_application (lno integer, lamtrequired money, lamtapproved money, l_date date)

The relationship is as follows: branch, customer, loan_application are related with **ternary relationship**. TERNARY (bid integer, cno integer, lno integer).

a) Write a function that returns the total number of customers of a particular branch.(Accept branch name as input parameter.)

b) Write a function to find the minimum loan amount approved.

c) Create a function which returns the total number of customers who have applied for a loan more than Rs.200000. (Accept loan amount as input parameter)

SET C

Student- Teacher database

Consider the following database maintained by a school. The school maintains information about students and the teachers. It also gives information of the subject taught by the teacher.

Following are the tables:

student (sno integer, s_name char(30), s_class char(10), s_addr char(50))

teacher (tno integer, t_name char (20), qualification char (15), experience integer)

The relationship is as follows: STUDENT-TEACHER**: Many to Many** with descriptive attribute SUBJECT.

a) Write a function to find name of the most experienced teacher for "Computer".

b) Write a function to find the teacher teaching maximum number of subjects.
c) Write a function to find the number of teachers having qualification "NET".

SET D

Project – Employee Database

Consider the following database

Project (pno int, pname char (30), ptype char (20), duration int)

Employee (empno int, ename char (20), joining_date date)

The relationship between Project and Employee is **many to many** with descriptive attribute start_date.

Create the above database in PostGreSQL and insert sufficient records.

Execute any two of the following using PL/pgSQL

a. Write a stored function to accept project type as an input and display all project names of that type.
b. Write a function which accepts employee name and prints the details of the project which the employee works on.
c. Write a function to accept project name as input and returns the number of employees working on the project.

SET E

Student - Subject Database

Consider the following database

Student (roll_no integer, name varchar(30), address varchar(50), class varchar(10)) Subject (scode varchar(10), subject_name varchar(20))

Student-Subject are related with **Many to Many** relationship with attributes marks_scored. Create the above database in PostGreSQL and insert sufficient records.

a. Write a function which will accept the name and print all the details of that student.
b. Write a function to accept student roll_no as input and displays details of that student.
c. Write a stored function using cursors, to accept class from the user and display the details of the students of that class.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Assignment 3: Cursors

PL/SQL Cursors provide a way to select multiple rows of data from the database and then to process each row individually. Using a cursor, we can traverse up and down a result set and retrieve only those rows which are explicitly requested. Cursors basically help an application to efficiently use a static result set.

Declaring Cursor Variables

All access to cursors in PL/pgSQL goes through cursor variables, which are always of the special data type refcursor. We can declare a cursor variable either as a bound cursor variable or an unbound cursor variable.

a. To create an unbound cursor variable, just declare it as a variable of type refcursor.

b. To create a bound cursor variable, use the following cursor declaration.

syntax

name CURSOR [(arguments)] FOR query;

where arguments, if specified, is a comma-separated list of pairs 'name datatype' that define names to be replaced by parameter values in the given query. The actual values to substitute for these names will be specified later, when the cursor is opened (parameterized cursors).

Opening Cursors

Before a cursor can be used to retrieve rows, it must be opened. PL/pgSQL has three forms of the OPEN statement, two of which use unbound cursor variables while the third uses a bound cursor variable.

Syntax for OPEN FOR query

OPEN unbound_cursorvar FOR query;

Syntax for OPEN FOR EXECUTE

OPEN unbound_cursorvar FOR EXECUTE query_string USING expression [, ...]];

Syntax for Opening a Bound Cursor

OPEN bound_cursorvar [(argument_values)];

Using Cursors

Once a cursor has been opened, it can be manipulated with the statements describedbelow.

FETCH

Syntax :

FETCH [direction { FROM | IN }] cursor INTO target;

The direction clause can be any of the following variants :

NEXT, PRIOR, FIRST, LAST, ABSOLUTE count, RELATIVE count, FORWARD, orBACKWARD. Default is NEXT.

MOVE : MOVE repositions a cursor without retrieving any data.

Syntax :

MOVE [direction { FROM | IN }] cursor;

The direction clause can be any of the variants NEXT, PRIOR, FIRST, LAST,

ABSOLUTEcount, RELATIVE count, ALL, FORWARD [count | ALL], or BACKWARD [count | ALL].

CLOSE : CLOSE closes the portal underlying an open cursor. This can be used to releaseresources earlier than end of transaction, or to free up the cursor variable to be openedagain.

Syntax :

CLOSE cursor;

Example 1:

Consider a relation, Employee (eno, ename, deptno,salary). We write a function, to Definea cursor to print the details of the employee along with commission earned for eachemployee. Commission is 20% of salary for employees of dept no = 5; its 50% of salary foremployees of dept no = 8; its 30% of salary for employees of dept no = 10.

```
Create function cursor_demo( ) returns integer as ‘
Declare
Emp–rec Employee%rowtype
C–emp cursor for Select * from employee; Comm
Number (6,2);
Begin
Loop

Fetch C–emp into emp–rec; If emp–rec.deptno
= 5 then
Comm:= emp–rec.salary * 0.2; Else if emp–rec.deptno = 8
then
Comm := emp–rec.salary * 0.5; Else If emp–rec.deptno = 10
then
Comm := emp–rec.salary * 0.3;
End if; End if; Endif;
Raise notice ‘’emp–rec.ename||emp–rec.deptno||emp–rec.salary||comm. ‘’; Exit when
not found;
End loop;
Close C–emp;
End; ‘ language ‘plpgsql’;
```

Example 2:

Consider the following relations, Dept(dno, dname) Employee(eno, dno, ename, salary) and the relation between Dept and Employee is one to many(1:M). Now we write a PL/pgSQL function to print the list of employees, department wise. Here we use 2 cursors, the 1st Cursor retrieves the department Information for each department and the 2nd cursor, to which dept number is sent as a parameter retrieves the employees for that department.

```
Create function parameterizedcursor_demo( ) returns integer as ‘
Declare
Begin
```

```
d–Info cursor for Select * from dept;
E–Info cursor (dnumdept.dno%type) for Select * From
Emplyee Where dno = dnum; X number := 0;

For drecIn d–Info Loop

Raise notice ‘’ drec.dno || drec.dname’’; For erecIn E–Info
(drec.dno)
Loop
Raise notice ‘’erec.eno || erec.ename || erec.salary’’; X := X + 1;
End loop;
Raise notice “Total employees for:||drec.dname||‘is %’|| X’’; End loop;
Return (X);
End; ‘ language ‘plpgsql’;
```

NOTE: Solve any three sets from the Following:

SET A

Bus – Route Database

Consider the following database

Bus (bus_no int, capacity int , depot_name varchar(20))

Route (route_no int, source varchar(20), destination varchar(20), No_of_stations int) Bus and Route are related with **many to many** relationship.

Create the above database in PostGreSQL and insert sufficient records.

a. Write a stored function using cursor, which will give details of all routes on which bus no 108 is running.
b. Write a stored function using cursor, which will give details of all buses on route from “Station” to “Airport”.

SET B

Student –Teacher database

Consider the following database

Teacher(t_no int, t_name varchar(20), age int, yr_experience int)

Subject (s_no int, s_namevarchar(15))

Teacher and Subject are related with **many to many** relationship

Create the above database in PostGreSQL and insert sufficient records.

a. Write a stored function using cursor which will accept the subject name and print the names of all teachers teaching that subject.
b. Write a cursor to accept the subject's name from the user as an input and display names of all teachers teaching that student.

SET C

Person - Area Database

Person (pno int, name varchar (20), birthdate date, income money) Area
(aid int, aname varchar (20), area_type varchar (5))
The person and area related to **many to one** relationship. The attribute 'area_type' can have values either 'urban' or 'rural'.
Create the above database in PostGreSQL and insert sufficient records.

a. Write a cursor to accept a month as an input parameter from the user and display the names of persons whose birthday falls in that particular month.
b. Write a cursor to display the names of persons living in urban area.
c. Write a cursor to print names of all persons having income between50,000 and 1,00,000.

SET D

Student – Competition Database

Consider the following entities and their relationship.
Student (s_reg_no int, s_name varchar(20), s_class varchar(20))
Competition (comp_no int, comp_name varchar(20), comp_type varchar(20))
Relationship between Student and Competition is **many-to-many** with descriptive attribute rank and year.

a) Write a cursor which will display year wise details of competitions held. (Use parameterized cursor)
b) Write a cursor which will display student wise total count of competition participated.

SET E

Car – Driver Database

Consider the following database:
Car (c_no int, owner varchar(20), model varchar(10), color varchar(10)
Driver (driver_no int, driver_namevarchar(20), license_no int, d_age int, salary float) Car
and Driver are related with **many to many** relationship
Create the above database in PostGreSQL and insert sufficient records.

a. Write a stored function with cursor which accepts the color and prints the names of all owners who own a car of that color.
b. Write a cursor which accepts the driver name and prints the details of all cars that this driver has driven, if the driver name is invalid, print an appropriate message.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Assignment 4: Handling errors and Exceptions

The RAISE statements raise errors and exceptions during a PL/pgSQL function execution. A Raise statement is also given the level of error it should raise and the string error message it should send to postgreSQL. The string can also be embedded with variables and expressions, that one needs to list along with the error message. The percent (%) sign is used as the place holder for the variables that are inserted into the string.

The syntax of the RAISE statement is as follows:

RAISE level ‘’message string ‘’ [, identifier [….]];

The three possible values for the RAISE statement’s level are as follows:

Value	**Explanation**
DEBUG	Debug level statements send the specified text as a debug message to the PostgreSQL log.
NOTICE	Notice level statements send the specified text as a Notice;
EXCEPTION	Exception level statements send the specified text as an ERROR. The exception level also causes the current transaction to be aborted.

Practice examples:

Example 1

In this example, the first raise statement gives a debug level message. The second and third raise statements send a notice to the user. The fourth raise statement displays an error and throws an exception, causing the function to end and the transaction to be aborted.

```
Create function raise_demo( ) returns integer as ‘
Declare
Int_var integer: =1;
Begin
-- raise a debug level message
Raise debug ‘’the raise demo function began’’; Int_var :=
int_var+1;
--raise a notice stating the change in value of variable
Raise notice ‘’ variable int_var‘s value is now % .’’,int_var;
--raise an exception
Raise exception ‘’variable % changed. Transaction aborted.’’,int_var;
Return 1;
End;
‘language ‘plpgsql’;
```

NOTE: Solve any three sets from the Following:

SET A

Project-Employee database

Consider the following database:

Project (pno int, pname char (30), ptype char (20), duration int)
Employee (empno int, ename char (20), joining_date date)
The relationship between Project and Employee is **many to many** with descriptive attribute start_date.

Create the above database in PostGreSQL and insert sufficient records.

a. Write a stored function to accept project name as input and print the names of employees working on the project. Also print the total number of employees working on that project. Raise an exception for an invalid project name.

b. Write a stored function to accept empno as an input parameter from the user and count the number of projects of a given employee. Raise an exception if the employee number is invalid.

SET B

Person – Area database

Person (pno int, name varchar (20), birthdate date, income money) Area
(aid int, aname varchar (20), area_type varchar (5))
The person and area related to **many to one** relationship. The attribute 'area_type' can have values either 'urban' or 'rural'.
Create the above database in PostGreSQL and insert sufficient records.

a. Write a stored function that accepts the area name as an input parameter from the user and displays the details of persons living in that area. Raise an exception if area name is invalid.

SET C

Wholesaler – Product database

Consider the following entities and their relationships.

Wholesaler (w_no, w_name, address, city)

Product (product_no, product_name, rate)

Relation between Wholesaler and Product is **Many to Many** with quantity as descriptive attribute.

Create the above database in PostGreSQL and insert sufficient records.

a. Write a function to accept quantity from user. Quantity must be within range 50-200. If user enters the quantity out of range, then raise a user defined exception "quantity_out_of _range" otherwise enter the record in table.
b. Write a function which accept rate from user. If user enters rate less than or equal to zero then raise an user defined exception "Invalid_Rate_Value" otherwise display message "Correct Input".
c. Write a function to accept product name as parameter. If entered product name is not valid then raise an user defined exception"Invalid_Product_Name" otherwise display product details of Specified product.

SET D

Student Teacher Database

Student (sno integer, s_name char(30), s_class char(10), s_addr Char(50))
Teacher (tno integer, t_name char (20), qualification char (15), experience integer)
The relationship is as follows:
Student-Teacher: **Many to Many** with descriptive attribute Subject.

a. Write a stored function to count the number of the teachers teaching to a student named " ". (Accept student name as input parameter). Raise an exception if student name does not exist.
b. Write a stored function to count the number of the students who are studying subject named " " (Accept subject name as input parameter). Display error message if subject name is not valid.
c. Write a stored function to display teacher details who have qualification as " " (Accept teacher's qualification as input parameter). Raise an exception for invalid qualification.

SET E

Railway Reservation System Database

TRAIN: (train_no int, train_name varchar(20), depart_time time , arrival_time time, source_stn varchar (20),dest_stn varchar (20), no_of_res_bogies int ,bogie_capacity int)
PASSENGER : (passenger_id int, passenger_name varchar(20), address varchar(30), age int ,gender char)
Relationships:
Train _Passenger: **Many to Many** relationship named ticket with descriptive attributes as follows
TICKET: (train_no int, passenger_id int, ticket_no int ,bogie_no int, no_of_berths int ,tdate date , ticket_amt decimal(7,2),status char)
Constraints: The status of a berth can be 'W' (waiting) or 'C' (confirmed).

a. Write a stored function to print the details of train wise confirmed bookings on date " " (Accept date as input parameter).Raise an error in case of invalid date.
b. Write a stored function to accept date and passenger name and display no of berths reserved and ticket amount paid by him. Raise exception if passenger name is invalid.
c. Write a stored function to display the ticket details of a train. (Accept train name as input parameter).Raise an exception in case of invalid train name.

Assignment Evaluation

0: Not Done [ ] 1: Incomplete [ ] 2: Late Complete [ ]

3: Needs Improvement [ ] 4: Complete [ ] 5: Well Done [ ]

Assignment 5: Triggers.

A trigger defines a function which occurs before or after, an action on a table. A trigger is implemented through PL/pgSQL, C or any other functional language that PostgreSQL can use to define a function. A trigger is a PL/pgSQL block that is associated with a table, stored in a database and executed in response to a specific data manipulation event. Triggers can be executed or fired in response to the following events.

a. A row is inserted into table
b. A row in a table is updated.
c. A row in a table is deleted.

Syntax for defining a database trigger

Create Trigger trigger–name
{Before | After} {event [or event …]} ON table–name for
each { Row | statement}
execute procedure function name (arguments);

A trigger procedure is created with the CREATE FUNCTION command, declaring it as a function with no arguments and a return type of trigger.

Special variables created automatically, on call to a trigger function, are as follows :

Variable Name	Purpose and contents
NEW	Holds the new database row for INSERT/UPDATE operations in row-level triggers
OLD	Holds the old database row for UPDATE/DELETE operations in row-level triggers
TG_NAME	Contains the name of the trigger actually fired.
TG_WHEN	Specifies the trigger timing. Contains a string of BEFORE or AFTER,
TG_LEVEL	Specifies the trigger type. Contains a string of either ROW or STATEMENT
TG_OP	Specifies the trigger operation. Contains a string of INSERT, UPDATE, or DELETE.
TG_RELID	Contains the object ID of the table that caused trigger invocation.
TG_TABLE_NAME	Contains the name of the table that caused the trigger invocation.
TG_TABLE_SCHEMA	Contains the name of the schema of the table that caused trigger invocation
TG_NARGS	Number of arguments given to the trigger procedure
TG_ARGV[ ]	Contains the arguments for the trigger statement.

Example 1

This trigger ensures that any time a row is inserted or updated in the table, the current user name and time are stamped into the row. And it checks that an employee's name is given and that the

salary is a positive value.

CREATE TABLE employee (ename text, esalary integer, last_date timestamp, last_usertext);

```
CREATE FUNCTION emp_timestamp() RETURNS trigger AS ‘
BEGIN
-- Check that empname and salary are given IF
NEW.ename IS NULL THEN
RAISE EXCEPTION 'empname cannot be null'; END IF;
IF NEW.esalary IS NULL THEN
RAISE EXCEPTION '% cannot have null salary', NEW.ename; END IF;
IF NEW.esalary< 0 THEN
RAISE EXCEPTION '% cannot have a negative salary', NEW.ename; END IF;
-- Remember who changed the payroll when
NEW.last_date := current_timestamp; NEW.last_user :=
current_user;
RETURN NEW;
END;
‘ LANGUAGE ‘plpgsql’;
CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON employee FOR
EACH ROW EXECUTE PROCEDURE emp_stamp();
```

NOTE: Solve any three sets from the Following:

SET A

Movie – Actor Database

Consider the following database

Movie (m_name varchar (25), release_year integer, budget money)

Actor (a_name varchar(30), role varchar(30), charges money, a_address varchar(30)) Movie and Actor are related with **many to many** relationship.

Create the above database in PostGreSQL and insert sufficient records.

a. Write a trigger which will be executed whenever an actor is deleted from the actor table, display appropriate message.
b. Write a trigger which will be executed whenever a movie is deleted from the movie table, display appropriate message.
c. Write a trigger which will be executed whenever insertion is made to the movie table. If the budget is less than 1,00,000 do not allow the insertion. Give appropriate message.

SET B

Doctor – Hospital Database

Consider the following database

Doctor (d_no int, d_name varchar(30), specialization varchar(35), charges int)

Hospital (h_no int, h_name varchar(20), city varchar(10))

Doctor and Hospital are related with **many to one** relationship.

Create the above database in PostGreSQL and insert sufficient records.

a. Write a trigger before insert/update on Doctor table. Raise exception if charges are <0.
b. Write a trigger that restricts insertion of charges value greater than 400.

SET C

Student – Subject database

Consider the following database :

Student (rollno integer, name varchar(30),city varchar(50),class varchar(10))

Subject(Scode varchar(10),subject name varchar(20))

Student and subject are related with **M-M** relationship with attributes marks_scored.

Create the above database in PostGreSQL and insert sufficient records

a. Write a trigger before insert/update the marks_scored. Raise exception if Marks are negative.
b. Write a trigger which is executed when insertion is made in the student-subject table. If marks_scored is less than 0, give appropriate message and do not allow the insertion.
c. Write a trigger which will prevent deleting students from 'Mumbai' city.

SET D

Customer – Account database

Consider the following database

Customer (cno integer, cname varchar(20), city varchar(20))

Account (a_no int, a_type varchar(10), opening_date date, balance money) Customer

and Account are related with **one to many** relationship

Create the above database in PostGreSQL and insert sufficient records.

a. Write a trigger which is executed whenever update is made to the account table. If the balance becomes less than 1000, print an error message that balance cannot be less than 1000.
b. Write a trigger before deleting an account record from Account table. Raise a notice and display the message "Account record is being deleted."
c. Write a trigger before inserting an account record in Account table and raise exception if balance is <500.

SET E

Project-Employee Database

Consider the following Entities and their Relationships for Project-Employee database.

Project (pno integer, pname char (30), ptype char (20), duration integer)

Employee (eno integer, ename char (20), qualification char (15), joining_date date)

Relationship between Project and Employee is **many to many** with descriptive attribute start_date date, no_of_hours_worked integer.

Constraints: Primary Key, pname should not be null.

Create trigger for the following:

a. Write a trigger before inserting into an employee table to check current date should be always greater than joining date. Display appropriate message.
b. Write a trigger before inserting into a project table to check duration should be always greater than zero. Display appropriate message.
c. Write a trigger before deleting an employee record from employee table. Raise a notice and display the message "Employee record is being deleted".

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]