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 III

WORKBOOK

Course Code: CS-203-MJ-P

(Lab Course based on CS-201-MJ-T & CS-202-MJ-T)

and

Course Code: CS-231-FP

(Field Project)

WORKBOOK

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

Data Structures I
and
Database Management Systems I

Course Type: Major Core
Course Code: CS-203-MJ-P
Course Title: Lab Course based on CS-201-MJ-T & CS-202-MJ-T

SEMESTER III

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-201-MJ-T & CS-202-MJ-T Data structures I and Database Management Systems I.

Workbook is divided in two sections.

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

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 practical syllabus is divided into six 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.

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	Searching Algorithms		
2	Sorting Algorithms(Non recursive) – Bubble, Insertion, Selection		
3	Sorting Algorithms(Recursive) – Counting, Merge, Quick		
4	Linked List(Singly and Doubly)		
5	Stack		
6	Queue		
a. Total out of 30			

Section II Database Management System I

Sr. No.	Assignment Name	Marks (out of 5)	Signature
1.	Data Definition Language commands (Create)- Create simple tables(All Constraints)		
2.	Data Definition Language commands (Create)- Create more than one table, with referential integrity constraint		
3.	Data Definition Language commands (Alter, Drop) Data Manipulation Language commands (Insert, Update and delete)		
4.	Simple queries		
5.	Queries with set operations		
6.	Nested Queries, Views		
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

Assignment 1

Searching Algorithms

Searching is the process of finding a value in a list of values. The searching algorithms you are to use in this assignment are linear search and binary search.

Linear search -

In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.

Algorithm:

Step 1: Start

Step 2: Accept n numbers in an array num and a number to be searched

Step 3: set i=0 and flag=0

Step 4: if i<n then goto next step else goto step 7

Step 5: Compare num[i] and number if equal then set flag=1 and goto step 7

Step 6: i=i+1 and goto step 4

Step 7: if (flag=1) then

print “Required number is found at location i+1”

else

print “Require data not found”

Step 8: Stop

Time Complexity:

Best Case: O(1) Worst Case: O(n) Average Case: O(n)

Binary Search -

Binary Search is used with sorted array or list. So a necessary condition for Binary search to work is that the list/array should be sorted. It works by repeatedly dividing in half the portion of the list that could contain the item.

Algorithm:

Step 1: Start

Step 2: Accept n numbers in an array num and a number to be searched

Step 3: set low=0, high=n-1 and flag=0

Step 5: if low <= high then

middle=(low+high)/2

else

goto step 8.

Step 6: if (num[middle]=number)

position=middle, flag=1 goto step 8

else if (number< num[middle])

high=middle-1

else

low=middle+1

Step 7: goto step 5
Step 8: if flag=1
print “Required number is found at location position+1”
else
print “Required number is not found.
Step 9: Stop

Time Complexity:

Best Case: O(1) Worst Case: O(log n)

Average Case: O(log n)

Random array creation using random (rand ()) function -

The data to be searched is in memory usually in an array. It could be an array of integers, characters, strings or of defined structure type. To test a searching algorithm, we require large data set. Data is generated using random (rand ()) function. The array of random integers in the range 0 to 99 is generated by using following code:

```
void generate ( int * a, int n)
{
    int i;
    for (i=0; i<n; i++)
        a[i]=rand()%100;
}
```

Set A

a) Create a random array of n integers. Accept a value x from user and use linear search algorithm to check whether the number is present in the array or not and output the position if the number is present.

b) Accept n sorted values in array from user. Accept a value x from user and use binary search algorithm to check whether the number is present in sorted array or not and output the position if the number is present.

Set B

a) Read the data from file 'student.txt' containing names of student and their class (FY, SY, TY). Accept a student name from user and use linear search algorithm to check whether the name is present in the file and output the Class of student otherwise output "Student not in the list".

b) Read the data from file 'student.txt' containing names of student and their class(FY, SY,TY). Accept a student name from user and use binary search algorithm to check whether the name is present in the file and output the class of student otherwise output "Student not in the list".

Set C

a) If the file contains multiple occurrences of a given element, linear search will give the position of the first occurrence, what modifications are required to get the last occurrence?

b) If the file contains multiple occurrences of a given element, will binary search output the position of first occurrence or last occurrence?

c) Which is best case search when searching using linear search and when using binary search?

d) What modifications are required to linear search and binary search algorithm to count the number of comparisons?

e) What modifications are required to binary search so that it returns the position where x can be inserted in the sorted array to retain the sorted order?

Assignment Evaluation

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

Practical In-charge

Assignment 2

Sorting Algorithms (Non Recursive) – Bubble Sort, Insertion Sort, Selection Sort

Arranging the data in an ordered sequence is called as sorting. In this assignment we are working on non-recursive sorting algorithms like bubble sort, insertion sort and selection sort.

Bubble Sort

Bubble sort is a simple sorting algorithm that repeatedly steps through the data, compares adjacent elements and swaps them if they are in the wrong order. The pass is repeated until the list is sorted.

Algorithm:

Step1: Start
Step2: Accept ‘n’ numbers in array ‘A’
Step3: set i=0
Step4: if i<n-1 then goto next step else goto step 9
Step4: set j=0
Step5: if j<n-i-1 then go to next step else goto step 8
Step6: if A[j] > A[j+1] then
interchange A[j] and A[j+1]
Step7: j=j+1 and goto step 5
Step8: i=i+1 and goto step 4
Step9: Stop

Time Complexity:

Best Case: O(n) Worst Case: $O(n^2)$ Average Case: $O(n^2)$

Insertion Sort

Insertion sort is a simple sorting algorithm that works by iteratively building a sorted array one element at a time. It is an in-place algorithm, meaning it sorts the array without requiring additional memory.

Algorithm:

Step1: Start
Step2: Accept ‘n’ numbers and store all in array ‘A’
Step3: set i=1
Step4: if i<=n-1 then goto next step else goto step 10
Step5: set Temp=A[i] and j=i-1
Step6: if Temp < A[j] && j>=0 then goto next step else goto step 9
Step7: set A[j+1]=A[j]
Step8: set j=j-1
Step9: set A[j+1]=Temp
Step10: Stop

Time Complexity:

Best Case: O(n) Worst Case: $O(n^2)$ Average Case: $O(n^2)$

Selection Sort:

Selection sort is a simple, in-place comparison-based sorting algorithm that divides the input data into a sorted and an unsorted manner. It repeatedly finds the minimum (or maximum) element from the unsorted portion and swaps it with the first element of the unsorted portion, thus extending the sorted region.

Algorithm:

Step1: Start

Step2: Accept ‘n’ numbers and store all in array ‘A’

Step3: set i=0

Step4: if i<n-1 then goto next step else goto step 11

Step5: set min=i and j=i+1

Step6: if j<n then goto next step else goto step 9

Step7: if A[j]<A[min] then

min=j

Step8: set j=j+1 and goto step 7

Step9: if (min not equal to i) then

interchange A[i] and A[min]

Step10: i=i+1 and goto step 4

Step11: Stop

Time Complexity:

Best Case: $O(n^2)$ Worst Case: $O(n^2)$ Average Case: $O(n^2)$

Set A

a) Sort a random array of n integers (accept the value of n from user) in ascending order by using bubble sort algorithm.

b) Sort a random array of n integers (create a random array of n integers) in ascending order by using insertion sort algorithm.

c) Sort a random array of n integers (accept the value of n from user) in ascending order by using selection sort algorithm.

Set B

a) Read the data from "Student.txt (Stud_name, Age, Percentage)" file and sort on Stud_name using bubble sort.(Create a separate sorted file).

b) Read the data from "Student.txt (Stud_name, Age, Percentage)" file and sort on Age using insertion sort. (Create a separate sorted file).

c) Read the data from "Student.txt (Stud_name, Age, Percentage)" file and sort on Percentage using selection sort.(Create a separate sorted file).

Set C

a) What modification is required to bubble sort, insertion sort and selection to sort the integers in descending order?

d) What modifications are required to bubble sort to count the number of swaps?

c) What modifications are required to insertion sort to count the number of key comparisons?

d) What modifications are required to output the array contents after every pass of the sorting algorithm?

Assignment Evaluation

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

Practical In-charge

Assignment 3

Sorting Algorithms –Counting Sort, Merge Sort, Quick Sort

Arranging the data in an ordered sequence is called as sorting. In this assignment we are working on recursive sorting algorithms like Counting Sort, Merge Sort, Quick Sort.

Counting Sort

It is a non-comparison-based sorting algorithm that sorts integers within a specific range. It works by counting the number of occurrences of each distinct element and using this information to place elements in their correct positions in the output array.

CountingSort (array, size)

max = find largest element in array

initialize count array with all zeros

for j <= 0 to size

find the total count of each unique element and store the count at j^{th} index in count array

for i<= 1 to max

find the cumulative sum and store it in count array itself

for j <= size down to 1

restore the elements to array

decrease count of each element restored by 1

Merge Sort:

Merge sort is a divide-and-conquer algorithm that sorts a list by dividing it into smaller subsists, sorting those sub lists and then merging them back together. It works by recursively splitting the list into halves until each sub list has only one element, which is inherently sorted. Then, it merges the sorted sub lists back together to produce the final sorted list.

Algorithm:

mergesort(a, N) algorithm:

Step1: Start

Step2: Accept N numbers and store into array a

Step3: Assign low=0 and high=N-1

Step4: if low < high

mid=(low+high)/2

mergesort(a, low,mid);

mergesort(a, mid+1, high);

merge(a,low,mid, high);

Algorithm:

Merge:

Step1: i=low, j=mid+1, k=0;

Step2: while i<=mid && j<=high

if(a[i]<=a[j])

b[k]=a[i] k=k+1, i=i+1

else

b[k]=a[j] k=k+1, j=j+1

Step3: while(i<=mid)
b[k]=a[i] k=k+1, i=i+1
Step4: while(j<=high)
b[k]=a[j] k=k+1, j=j+1
Step5: j=low, k=0
Step6: while j<=high
a[j]=b[k] j=j+1, k=k+1
Step7: stop

Quick Sort:

Quick sort is a highly efficient divide-and-conquer sorting algorithm. It works by selecting a 'pivot' element from the list and partitioning the other elements into two sub-arrays: those less than the pivot and those greater than the pivot.

Algorithm:

Quicksort(A,n)

Step1: Start
Step2: Accept ‘n’ numbers and store all in array ‘A’
Step3: lb=0, ub=n-1
Step4: if (lb<ub)
j=Partition(A,lb,ub)
Quicksort(A,lb,j-1)
Quicksort(A,j+1,ub)
Step5: Stop

Partition (A, lb, ub)

Step1: down=lb, up=ub
Step2: pivot=A[lb]
Step3: while (A[down]<=pivot && down<up)
down++
Step4: while (A[up]>pivot && up>down)
up--
Step5: if (down < up)
Interchange A[down] and A[up] and goto step 3.
Step6: Interchange A[up] and pivot
Step7: return up
Step8: Stop

Set A

a) Accept the array of n integers from user and sort the array in ascending order by using recursive Counting sort algorithm.

b) Create random array of n integers and sort the array in ascending order by using recursive Merge sort algorithm

c) Accept the array of n integers from user and sort the array in ascending order by using recursive Quick sort algorithm.

Set B

a) Read the data from the ‘employee.txt’ file and sort on age using Counting sort and Merge sort. Write the sorted data to another file 'sortedemponage.txt'.

b) Read the data from the file and sort on names in alphabetical order (use strcmp) using Quick sort and write the sorted data to another file 'sortedemponname.txt'.

Set C

a) What modifications are required to choose the pivot element randomly instead of choosing the first element as pivot element while partitioning in Quick sort algorithm?

b) Compare the system time taken by Merge sort and bubble sort by using time command on a random array of integers of size 10000 or more.

c) What modification is required to Merge sort to sort the integers in descending order?

d) In 'employee.txt' there are records with same name but different age and salary values. What are the relative positions when the data is sorted on name using Merge sort and what happens in case of quick sort?

e) Sort a random array of integers of large size and store the sorted file. Compare the system time taken by Quicksort on a random file of large size and the sorted file of same size. Repeat the same for Merge sort. Does sorted file give best time?

Assignment Evaluation

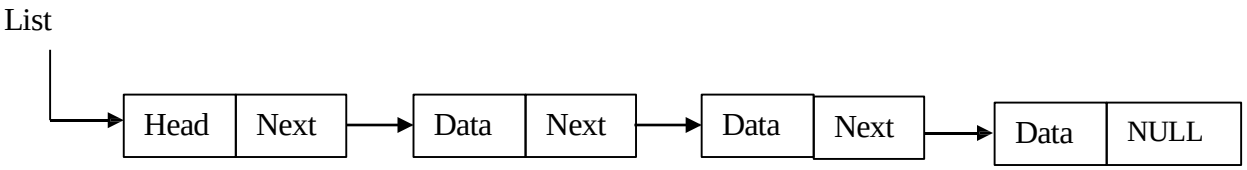
0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 4

Singly Linked List

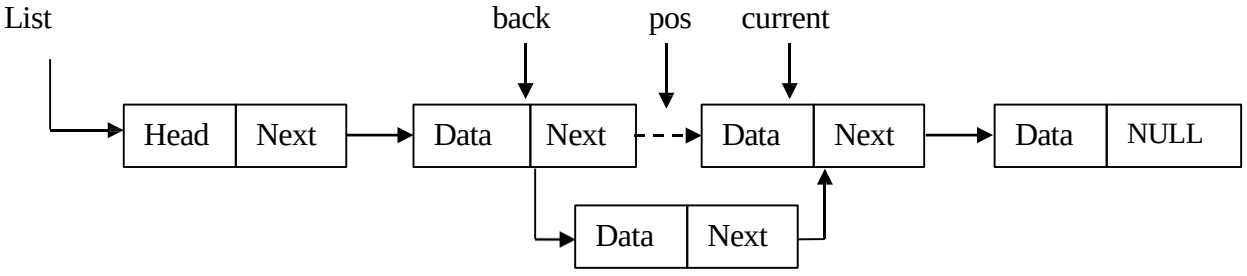
A singly linked list is a fundamental data structure in computer science, where data elements (nodes) are linked together in a sequence, and each node points to the next node in the sequence. The last node in the list points to null, indicating the end. This allows for dynamic addition and removal of elements without rearranging the entire data structure, making it a versatile choice for various applications.



Operations on Linked List:

append(L, x)	Insert the data element x by creating the node containing the data element x and inserting it in last position.
insert (L, x, pos)	inserts the data element x by creating the node containing the data element x and inserting it in position pos. The links are appropriately changed. If pos equals 1, the node is inserted in first position immediately after the header. If pos is greater than the nodes present in the list, the node is added at the end of the list.
search (L, x)	Searches for the data element x and returns the pointer to the node containing x if x is present or returns NULL.
delete (L, x)	Deletes the node containing the data element x by appropriately modifying the links.
delete (L, pos)	Delete the node at a position given by pos. If the position pos is invalid then display appropriate message.
display (L)	Displays all the data elements in the list

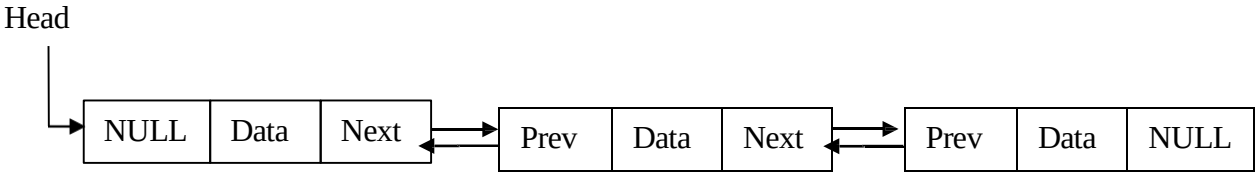
In a singly linked list there is only a link to the next element. For insertion as well as deletion one need to traverse with two pointers back and current.



Doubly Linked List

In a Doubly linked list there is link to the next element as well as a link to the previous element. For insertion one needs only the pointer to current element and same is true for deletion.

The two links allow traversal of list in both directions i.e. forward and backward. It is easy to reverse doubly linked list as compared to singly linked list.



Creation of Custom Header File / Library file

Header files

Header files are helping file of C program which holds the definitions of various functions and their associated variables. These header files are imported into the C program with the help of pre-processor *#include* statement.

The basic syntax of using these header files is:

#include<file>/* This syntax searches for file-name in the standard list of system directories */
OR

#include "file"/* This technique is used to search the file(s) within the directory that contains the file where this include statement is written. */

The **custom header file** [.h extension] is a file that contains user defined functions. The Function should accept input (arguments) and generate the output after processing the received input.

For example, header file [arithmetic] has following functions

- Here, functions only accept value and returns the result after processing the input. Functions written inside header file should not have any scanf / gets or printf / puts statements.

```
int add (int a.int b)
{
   return(a+b);
}
int mult(int a,int b)
{
     return(a*b);
}
```

Working of functions in header file in C:

- Let's try to explore math.h header file.
- This header file contains the function called as sqrt.
- The signature of the sqrt function is double sqrt(double arg)
- The sqrt function accepts one argument and returns the result. It doesn't directly print the result inside sqrt function.

Implement a list library (singlylist.h) for a singly linked list with Append, Insert, Search and Delete operations.

Set A

a) Write a menu driven program for the following operations.
   i. Insert an element in a linked list in a particular position
   ii. Search an element in a linked list.
   iii. Delete a particular element from a linked list.

b) Write a menu driven program for the following operations.
   i. Reverse a linked list and display it.
   ii. Accept an element from user and Concatenate it with a created link list.
   iii. Merge two linked list and display it.

Set B

Implement a list library (doublylist.h) for a doubly linked list with Append, Insert, Search and Delete operations.

a) Write a menu driven program for the following operations.
   i. Insert an element in a linked list in a particular position
   ii. Search an element in a linked list.
   iii. Delete a particular element from a linked list.

b) Write a menu driven program for the following operations.
   i. Reverse a linked list and display it.
   ii. Accept an element from user and Concatenate it with a created link list.
   iii. Merge two linked list and display it.

c) Write a program that adds two single variable polynomials. Each polynomial should be represented as a list with linked list implementation.

Set C

a) How to divide a singly linked list into two almost equal size lists?

b) The union operation of two disjoint sets takes two disjoint sets S1 and S2 and returns a disjoint set S consisting of all the elements of S1 and S2 and the original sets S1 and s2 are destroyed by the union operation. How to implement union in O(1) time using a suitable list data structure for representing a set?

c) Which operation is performed more efficiently in doubly linked list than singly linked list?

d) What is difference between singly circular linked list and doubly circular linked list?

e) What is the method to reverse a singly linked list in just one traversal?

Assignment Evaluation

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

Practical In-charge

Assignment 5

Stack

A stack is a linear data structure in which elements are added and removed only from one end, which is called the top. Hence, a stack is called a LIFO (Last-In, First-Out) data structure as the element that is inserted last is the first one to be taken out.

1. Static Implementation:

Stack is a simple data structure that follows the Last In, First Out (LIFO) principle. It is akin to a stack of browser tabs: new ones open on top, and when closing, it is always the most recent one.

Stacks operate through a single point using a push/pop mechanism. The "top" is where all adding (push) and removing (pop) happens. Thus, stacks are efficient and easy to manage.

The basic operations to be performed on a stack are:

Init(S)	Create an empty stack and initializes value of top to -1 indicating the stack is empty
Push(S,x)	Insert element x into stack S
Pop(S)	Deletes and returns topmost element from stack S
Peek(S)	Returns topmost element from the stack S without deleting element.
isEmpty(S)	Checks and returns if the stack S is empty or not. Stack isEmpty when top==-1
isFull(S)	Checks and returns if the stack S is full or not. Stack isFull when top==MAX-1

2. Dynamic Implementation

A stack is implemented dynamically by using a Linked list where each node in the linked list has two parts, the data element and the pointer to the next element of the stack. Stack is a single entity i.e. a pointer pointing to the top node in the linked list. The stack elements can be integers, characters, strings or user defined types. Dynamic lists can grow without limits.

The operations to be performed on a stack are

Init(S)	Create an empty stack and initializing top to NULL indicating the stack is empty
S=Push(S,x)	Adding an element x to the stack S requires creation of node containing x and putting it in front of the top node pointed by S. This changes the top node S and the function should return the changed value of S.
S=Pop(S)	Deletes top node from stack and returns. Next element will become top of stack and function returns the changed value of S.
X=Peek(S)	Returns topmost element from the stack S without deleting element.

Applications of Stack

Stack can be used in many applications like expression conversion, evaluation, parenthesis checking, checking string is palindrome or not etc. Infix, prefix, and postfix notations are three different but equivalent notations of writing algebraic expressions. Reversing string of characters, converting the infix notation into postfix notation and evaluating the postfix notation make extensive use of stacks as the primary tool. Following are few applications which extensively uses stack for their implementation.

1) String reverse and checking palindrome string:

A string of characters can be reversed by reading each character from a string starting from the first index and pushing it on a stack. Once all the characters have been read, the characters can be popped one at a time from the stack and then stored in the another string starting from the first index.

Algorithm to reverse the string:

1. Read the string character by character.
2. Push every character into the stack of characters.
3. When string becomes empty pop every character from stack and attach to the new string.

2) Infix to Postfix conversion:

Infix, prefix, and postfix notations are three different but equivalent notations of writing algebraic expressions. In postfix notation, operators are placed after the operands, whereas in prefix notation, operators are placed before the operands. While expression conversion precedence of operators is considered.

The precedence of operators can be given as follows:

Highest priority: ^ or $ (Exponential operator)
Higher priority: *, /, %
Lower priority: +, –

The algorithm accepts an infix expression that may contain operators. The algorithm uses a stack to temporarily hold operators. The postfix expression is obtained from left-to-right using the operands from the infix expression and the operators which are removed from the stack.

Algorithm for infix to postfix expression conversion:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else, If the precedence of the scanned operator is greater than the precedence of the operator in the stack (or the stack is empty or the stack contains a ‘(‘), push it.
4. Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
5. If the scanned character is an ‘(‘, push it to the stack.

6. If the scanned character is an ‘)’, pop the stack and output it until a ‘(‘is encountered, and discard both the parenthesis.
7. Repeat steps step 2 to step 7 until infix expression is scanned.
8. Print the output.
9. Pop the element form stack and print the output until stack it is not empty.

3) Evaluation of postfix string:

Postfix notations are evaluated using stacks. Every character of the postfix expression is scanned from left to right. If the character is an operand, it is pushed onto the stack. Else, if it is an operator, then the top two values are popped from the stack and the operator is applied on these values. The result is then pushed onto the stack.

Algorithm for evaluation of postfix expressions:

1) Create a stack to store operands (or values).
2) Scan the given expression and do following for every scanned element.
   2.1 If the element is a variable then, push its value into the stack
   2.2 If the element is an operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack
3) When the expression is ended, pop element from stack which is the final answer.

Set A

a) Perform the following operations using static implementation of stack
   i. Accept n integers from user and push it into the stack.
   ii. Pop the element from the stack and display it (Use isEmpty())

b) Perform the following operations using dynamic implementation of stack
   i. Accept n integers from user and push it into the stack.
   ii. Pop the element from the stack and display it.

Set B

a) Perform the following operations using implementation of stack (Static or dynamic)
   i. Accept n integers from user and push it into the stack.
   ii. Reverses a string of characters using stack and check weather given string is palindrome or not.
b) Write a menu driven program to perform the following operations on stack
   i. Convert an infix expression of the form (a*(b-c*d)/((a+d)/b)) into its equivalent postfix notation.
   ii. Evaluation of postfix expression

Set C

a) In dynamic implementation of stack, how to modify pop operation so that it also returns the popped element as an argument of the pop function?

Assignment Evaluation

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

Practical In-charge

Assignment 6
Queue

Queue:

A queue is a linear data structure which follows a particular order in which the operations are performed. The order is First in First out (FIFO) or Last in Last out (LILO).

In queue elements are added from one end called rear and removed from other end called front. Linear queue can be implemented statically using arrays and dynamically using linked lists.

1) Static Implementation of linear queue:

A Queue is implemented statically by using an array of size MAX to hold elements and two integers called front and rear.

A queue is a single entity that is a structure made up of the array, rear and front. Elements are added from rear end of the queue and can be deleted from front end of the queue. The 'front' stores the position of the current front element and 'rear' stores the position of the current rear element of the queue. The Queue elements can be integers, characters, strings or user defined types.

The basic operations to be performed on a Queue are:

init (Q)	Create an empty queue by initializing both front and rear to -1 indicating the queue is empty.
add (Q, x)	adds an element x to the rear end of the queue Q
X=delete (Q)	Deletes an element x from front end of the queue Q.
X=peek (Q)	Without deleting ,returns the front element from the queue Q
isEmpty (Q)	Checks whether the queue is empty. Queue becomes empty when rear equals to front.
isFull(Q)	Checks whether the queue is full. Queue becomes full when front reaches to MAX -1.

2. Dynamic Implementation of linear queue:

A Queue is implemented dynamically by using a Linked list. Each node in the linked list has two parts, the data element and the pointer to the next element of the queue. Since Queue should be a single entity, we need to use only one external pointer while here we need two one for rear and one to the front. To avoid this, we use a circular linked list and Queue pointer is pointing to the rear of the queue. Front can be easily accessed as it is next to rear. The Queue elements can be integers, characters, strings or user defined types. There is no restriction on how big the Queue can grow.

The operations to be performed on a Queue:

init (Q)	Create an empty queue as a circular linked list by initializing S to NULL indicating that the queue is empty
add (Q, x)	Adding an element x to the queue Q requires creation of node containing x and putting it next to the rear and rear points to the newly added element. This changes the rear pointer Q and the function should return the changed value of Q. The function call will be as follows
X=delete (Q)	Deletes the front node from the queue Q which is actually next element to the rear pointer Q. However, if queue contains only one element, (Q->next == Q) then deleting this single element can be achieved by making empty Q (Q =NULL). Since the rear pointer Q is changed in this case, function should return the changed value of Q. The function call will be as follows Q=delete(Q);
X=peek (Q)	returns the data element in the front (Q->next) node of the Queue Q
isEmpty (Q)	Check if the Queue is empty which is equivalent to checking if Q==NULL
isFull(Q)	Checks whether the queue is full. Queue becomes full when front reaches to MAX -1.

1) Circular queue:

In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we cannot insert the next element even if there is a space in front of queue. Even though there is space available, the overflow condition still exists because the condition REAR = MAX – 1 still holds true. This is a major drawback of a linear queue.

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First in First Out) principle and the last position is connected back to the first position to make a circle, hence making the queue behave like a circular data structure or Ring Buffer.

In case of a circular queue, head pointer will always point to the front of the queue, and tail pointer will always point to the end of the queue.

Initially, the head and the tail pointers will be pointing to the same location, this would mean that the queue is empty. New data is always added to the location pointed by the tail pointer, and once the data is added, tail pointer is incremented to point to the next available location. In a circular queue, data is not actually removed from the queue. Only the head pointer is incremented by one position when **delete queue ()** is executed. As the queue data is only the data between head and tail, hence the data left outside is not a part of the queue anymore, hence removed. The head and the tail pointer will get reinitialized to **0** every time they reach the end of the queue. Also, the head and the tail pointers can cross each other. In other words, head pointer can be greater than the tail. Sounds odd? This will happen when we delete queue the queue a couple of times and the tail pointer gets reinitialized upon reaching the end of the queue.

init (Q)	Create an empty queue as a circular linked list by initializing S to NULL indicating that the queue is empty
AddQueue(Q, x)	This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position. **Steps:** 1. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)). 2. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 &&front != 0) if it is true then set rear=0 and insert element.
X=DeleteQueue(Q)	This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position. **Steps:** 1. Check whether queue is Empty means check (front==-1). 2. If it is empty, then display Queue is empty. If queue is not empty, then step 3 3. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.
X=peek (Q)	returns the data element in the front (Q->next) node of the Queue Q
isEmpty (Q)	Check if the Queue is empty which is equivalent to checking if Q==NULL

Priority queue:

A priority queue is a data structure in which each element is assigned a priority. The priority of the element will be used to determine the order in which the elements will be processed.

The general rules of processing the elements of a priority queue are:

1. An element with higher priority is processed before an element with a lower priority.
2. Two elements with the same priority are processed on a first-come-first-served (FCFS) basis.
3. A Priority Queue is different from a normal queue, because instead of being a "first-in-first-out", values come out in order by priority, the highest-priority is retrieved first. The priority of the element can be set based on various factors. Priority queues are widely used in operating systems to execute the highest priority process first.

Set A

a) Perform the following operations using static implementation of Queue

i. Create a queue of n integers.

ii. Delete the element from the queue and display it.

b) Perform the following operations using dynamic implementation of Queue

i. Insert n elements in a queue.

ii. Delete the element from the queue and display it.

Set B

a) Perform the following operations on circular Queue (static or dynamic implementation)

i. Create a queue of n integers.

ii. Delete the element from the queue and display it.

b) Write a program to create a Priority Queue and display it.

i. Add an element with its priority into the queue.

ii. Delete an element from queue according to its priority.

Set C

a) Implement queue library using array or linked list. Use this queue library to simulate waiting list operations of railway reservation system.

Assignment Evaluation

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

Practical In-charge

Section-II

Database Management System I

Postgresql commands:

Debian based systems like Ubuntu:

Connect/login as root -

user@user-pc:~$ sudo -i -u postgres
postgres@user-pc:~$ psql
psql (9.3.5, server 9.3.6) Type "help" for help.

Redhat based systems like Centos / Fedora :

Connect/login as root -
Prompt_name$ su - postgres
Prompt_name$ initdb Database_name;
Prompt_name$ pg_ctl –D database name –l logfile start
Prompt_name$ createdb Database_name;
Prompt_name$ psql Database_name;

Assignment no.1

Data Definition Query (Create simple tables with all constraints)

To create simple tables, with the primary key constraint, tables with Check constraint, Unique constraint, not null constraint (include all data types)

You should read following topics before starting this exercise

1. Designing relations into tables
2. Using DDL statements to create tables
3. Different integrity constraints
4. Specification of different integrity constraints, in create table statement.

A table is a database object that holds data. A table must have unique name via which it can be referred. A table is made up of columns. Each column in the table must be given a unique name within that table. Each column will also have size, data-type and an optional constraint.

The data types permitted are

Data type	Syntax	Description	Example
Character data types	char(n)	It is fixed length character string of size n, default value 1 byte if n is not specified.	account_type char(6)
	varchar(n)	It is variable length character string with maximum size n.	employee_name varchar(50)
	Text	It is used to store large text data, no need to define a maximum	work_experience text
Numeric data types	Integer, int, serial	Serial is same as int, only that values are incremented automatically	Eno int, Eno serial
	Numeric	A real number with P digits, S of them after decimal point.	Sal numeric(5,2) Sal numeric(n)
	Float	Real number	Weight Float
Date and time type	Date	Stores date information	Birthdate date
	Time	Stores time information	Birthtime time
	Timestamp	Stores a date & time	Birth timestamp
Boolean and Binary type	Boolean, bool	Stores only 2 values : true or false, 1 or 0, yes or no, y or n, t or f	Flag Boolean

Constraints can be defined as either of the following:

Name	Description	Example
Column level	When data constraint is defined only with respect to one column & hence defined after the column definition, it is called as column level constraint	create table tablename (attribute1 datatype primary key , attribute2 datatype constraint constraint-name ……)
Table Level	When data constraint spans across multiple columns & hence defined after defining all the table columns when creating or altering a table structure, it is called as table level constraint	Create table tablename (attribute1 datatype , attribute2 datatype2 ,……, constraint pkey primary key(attribute1,atrribute2))

Syntax for **table creation:**

create table tablename (attribute list);

Attribute list : ([attribute name data type optional constraint] ,.....)

Primary key constraint:

Description	Properties	Example
A primary key is made up of one or more columns in a table, that uniquely identify each row in the table.	A column defined as a primary key, must adapt the following properties: a) The column cannot have NULL values. b) The data held across the column MUST be UNIQUE.	create table tablename (attribute1 datatype primary key, attribute2 datatype ……) create table tablename (attribute1 datatype, attribute2 datatype ...…, constraint pkey primary key(attribute1)) create table tablename (attribute1 datatype, attribute2 datatype ……, constraint constraint_name primarykey(attribute1,attribute2))

The following is the list of additional integrity constraints.

Constraint	Use	Syntax and Example
NULL	Specifies that the column can contain null values	create table client_master (client_no text NOT NULL, name text NOT NULL, addr text NULL, bal_due integer));
NOT NULL	Specifies that the column can not contain null values	create table client_master (client_no text NOT NULL, name text NOT NULL, addr text NOT NULL , bal_due integer));
UNIQUE	Forces the column to have unique values	create table client_master (client_no text UNIQUE,

		name text, addr text, bal_due integer));
CHECK	Specifies a condition that each row in the table must satisfy. Condition is specified as a logical expression that evaluates either TRUE or FALSE.	create table client_master (client_no text CHECK(client_no like 'C%'), name text CHECK(name=upper(name)), addr text));

SET A

1. Create a table with following details

Table Name	Player		
Columns	Column Name	Column Data Type	Constraints
1	player_id	integer	Primary key
2	Name	varchar (50)	
3	Birth_date	date	
4	Birth_place	varchar(100)	
Table level constraint	Name should not be NULL		

2. Create a table with following details

Table Name	Student		
Columns	Column Name	Column Data Type	Constraints
1	Roll_no	integer	
2	Class	varchar (20)	
3	Weight	numeric (6,2)	
4	Height	numeric (6,2)	
Table level constraint	Roll_no and class as primary key		

Set B

1. Create a table with details as given below

Table Name	Machine		
Columns	Column Name	Column Data Type	Constraints
1	Machine_id	integer	primary key
2	Machine_name	varchar (50)	NOT NULL, uppercase
3	Machine_type	varchar(10)	Type in ('drilling', 'milling', 'lathe', 'turning', 'grinding')
4	Machine_price	float	Greater than zero

5	Machine_cost	float	
Table level constraint	Machine_cost less than Machine_price		

2. Create a table with details as given below

Table Name	Employee		
Columns	Column Name	Column Data Type	Constraints
1	Employee_id	integer	Primary key
2	Employee_name	varchar (50)	NOT NULL, uppercase
3	Employee_desg	varchar(10)	Designation in (‘Manager’, ‘staff’, ‘worker’)
4	Employee_sal	float	Greater than Zero
5	Employee_uid	text	Unique
Table level constraint	Employee_uid not equal to Employee_id		

Set C

1. Create a table with details as given below

Table Name	Student		
Columns	Column Name	Column Data Type	Constraints
1	Stud_id	integer	Primary key
2	Stud _name	varchar (50)	NOT NULL, uppercase
3	Stud _Class	varchar(10)	Class in (‘FY’, ‘SY’, ‘TY’)
4	Stud _Marks	float	Greater than Zero
5	Stud _uid	text	Unique
Table level constraint	Stud_uid not equal to Stud_id		

Assignment Evaluation

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

Practical In-charge

Assignment no.2

Data Definition Query (Create simple tables with referential integrity constraint).

To create more than one table, with referential integrity constraint, PK constraint.

You should read following topics before starting this exercise

1. Referential Integrity constraints, relationship types (1-1,1-m, m-1, m-m)
2. Handling relationships while converting relations into tables in Relational Database, so that there are no redundancies.

The integrity constraints help us to enforce business rules on data in the database. Once an integrity constraint is specified for a table or a set of tables, all data in the table always follows the rule specified by the integrity constraint.

Referential integrity constraints designate a column or grouping of columns in a table called child table as a foreign key and establishes a relationship between that foreign key and specified primary key of another table called parent table.

The following is the list of constraints that can be defined for enforcing the referential integrity constraint.

Constraint	**Use**	**Syntax and Example**
Primary key	Designates a column or combination of columns as primary key	PRIMARY KEY (column_name [,column_name])
Foreign key	Designates a column or grouping of columns as a foreign key with a table constraint	FOREIGN KEY (column_name [,column_name])
References	Identifies the primary key that is referenced by a foreign key. If only parent table name is specified it automatically references primary key of parent table	column_name datatype REFERENCES table_name[(column_name [,column_name])
On delete Cascade	The referential integrity is maintained by automatically removing dependent foreign key values when primary key is deleted	column_name datatype REFERENCES table_name[(column_name)][ON DELETE CASCADE]
On update set null	If set, makes the foreign key column NULL, whenever there is a change in the primary key value of the referred table	Constraint name foreign key column_name references referred-table(column_name) on update set null

Rules to Handle relationships, attributes, enhanced E-R concepts during table creation

Name	Description	Handling	Example	Create statement
One-to-one	A member from an entity set is connected to at most one member from the other entity set & vice-versa	The key attribute from anyone entity set goes to the other entity set (may be the entity set that has full participation in relation) , as a foreign key.	room and guest. room no is foreign key in guest relation. guest has full participation in relation.	create table room(rno int primary key, desc char(30)); create table guest(gno int, name varchar(20), rno int references room unique);
One-to-many, many-to-one	A member from the entity set on the one side is connected to one or more members from other entity set, but a member from the entity set on the many side , is connected to atmost one member of the entity set on one side.	The key attribute of the entity set on one side is put as foreign key in the entity set on the many side.	department and employee. Here department is on the one side & employee is on the many side.	create table dept(dno int primary key, dname char(20)); create table emp(eno int primary key, name char(30), dno int references dept));
May-to-many	A member from one entity set connected to one or more members of the other entity set & vice-versa	A new relation is created that will contain the key attributes of both the participating entity sets.	student and subject a student can opt for many subjects and a subject has many students opting for it.	create table student(sno int primary key, name varchar(20)); create table subject(sbno int primary key, name varchar(20)); create table st_sub(sno int references student, sbno int references subject, constraint pkey primary key(sno,sbno));
A multivalued attribute	an attribute having multiple values for each member of the entity set	A new relation is created, which will contain a place holder for the multivalued attribute and the key attributes of the entity set that	An employee having multiple contact numbers, like home phone, mobile number, office number etc. phone-no attribute in employee	create table emp(eno int primary key, name char(30)); create table emp_ph(eno int references emp, phno int ,

		contains the multivalued attribute	relation becomes a multivalued attribute.	constraint pkey primary key(eno, phno));
A multivalued, composite attribute	A composite attribute having multiple values , for each member of the entity set	A new relation is created, which will contain a place holder for each part of the composite attribute and the key attributes of the entity set that contains the composite multivalued attribute	An employee having multiple addresses, where each address is made up of a block no, street no, city, state. Hence the address attribute becomes a composite multivalued attribute.	create table emp(eno int primary key, name char(30)); create table emp_add(eno int references emp, addno int, hno int, street char(20), city char(20), constraint pkey primary key(eno,addno));
Generalization /specialization	The members of an entity set can be grouped into several subgroups, based on an attribute/s value/s. Each subgroup becomes an entity set. Depicts a parent-child type of relationship.	New relations for each subgroup, if the subgroups have its own attributes, other than the parent attributes. The parent entity set's key is added to each subgroup. If no specific attributes for each subgroup, then only the parent relation is created.	A person (parent entity set) can be an employee, a student, a retired person. Here employee has its own set of attributes like company, salary etc. a student has its own set of attributes like college/ school, course etc. a retired person has its own set of attributes like hobby, pension etc. so we create a person relation, a student relation, an employee, a retire person. The student, employee, retired person entity sets will have the key of the person entity set added to it.	create table person (ssno int primary key, name char(30)); create table emp(ssno int references person, eno int, cname char(20), sal float, primary key(ssno)); create table student(ssno int references person, class char(10), school varchar(50), primary key(ssno));

Set A

Create tables for the information given below by giving appropriate integrity constraints as specified.

1. Create the following tables:

Table Name	Property		
Columns	Column Name	Column Data Type	Constraints
1	P-number	integer	Primary key
2	Description	varchar (50)	Not null
3	Area	char(10)	

Table Name	Owner		
Columns	Column Name	Column Data Type	Constraints
1	Owner-name	varchar(50)	Primary key
2	Address	varchar (50)	
3	Phoneno	integer	

Relationship - A one-many relationships between owner and property. Define reference keys accordingly.

2. Create the following tables:

Table Name	Hospital		
Columns	Column Name	Column Data Type	Constraints
1	Hno	integer	Primary key
2	Name	varchar (50)	Not null
3	City	char(10)	

Table Name	Doctor		
Columns	Column Name	Column Data Type	Constraints
1	Dno	integer	Primary key
2	Dname	varchar (50)	
3	City	char(10)	

Relationship - A many-many relationships between hospital and doctor.

3. Create the following tables:

Table Name	Patient		
Columns	Column Name	Column Data Type	Constraints
1	Pno	integer	Primary key
2	Name	varchar (50)	Not null
3	Address	varchar(50)	

Table Name	Bed		
Columns	Column Name	Column Data Type	Constraints
1	Bedno	integer	Primary key
2	Roomno	integer	Primary key
3	Description	varchar(50)	

Relationship - a one–to-one relationship between Patient & Bed.

Set B

1. Create the following tables:

Table Name	Student		
Columns	Column Name	Column Data Type	Constraints
1	Sno	integer	Primary key
2	Name	varchar (50)	Not null
3	Address	varchar(50)	
4	Class	varchar(10)	

Table Name	Teacher		
Columns	Column Name	Column Data Type	Constraints
1	Tno	integer	Primary key
2	Tname	integer	Primary key
3	Qualification	varchar(50)	
4	TotalExperience	float	
5	Salary	float	Should be greater than 0

Relationship - A Many–to-Many relationships between Student and Teacher with descriptive attribute Marks.

2. Create the following tables:

Table Name	Project		
Columns	Column Name	Column Data Type	Constraints
1	Pno	integer	Primary key
2	Pname	varchar (30)	Not null
3	Ptype	char(20)	
4	Duration	integer	

Table Name	Employee		
Columns	Column Name	Column Data Type	Constraints
1	Eno	integer	Primary key
2	Ename	varchar(20)	
3	Qualification	varchar(50)	
4	Join_Date	date	
5	Salary	float	Should be greater than 0

Relationship -A Many–to-Many relationships between Project and Employee with descriptive attribute Start date(date), no_of_hours_worked(integer).

Set C

Create table for the information given below by choosing appropriate data types and integrity constraints as specified.

1. Table _____(_____, _____, _____, _____)
_____(_____, _____, _____, _____)

Constraints: _____, _____

2. Table _____(_____, _____, _____, _____)
_____(_____, _____, _____, _____)

Constraints: _____, _____

Relationship - _____

Instructor should fill in the blanks with appropriate values.

Assignment Evaluation

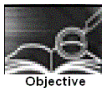
0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment no.3

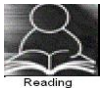
Data Definition Query (Alter, Drop)

Data Manipulation Language commands (Insert, Update and delete)



1. To drop a table from the database, to alter the schema of a table in the Database.

2. To insert / update / delete records using tables created in previous Assignments. (use simple forms of insert / update / delete statements)



You should read following topics before starting this exercise

1. Read through the drop, Alter DDL statement
2. Read through the insert, update, delete statement.
3. Go through the variations in syntaxes of the above statements.
4. Go through some examples of these statements
5. Go through the relationship types & conversion of relations to tables in RDB.
6. Normal forms 1NF, 2NF, 3NF

The Drop & Alter DDL statements:

Name	Description	Syntax	Example
Drop	Deletes an object (table/view/constraint) schema from the database	drop table table_name;	drop table employee; drop table sales; drop constraint pkey;
Alter	Alter table command of SQL is used to modify the structure of the table It can be used for following purposes a) adding new column b) modifying existing columns c) add an integrity constraint d) To redefine a column Restrictions on the alter table are that, the following tasks cannot be performed with this clause a) Change the name of the column b) Drop a column c) Decrease the size of a column if table data exists	alter table table_name add (new column_name datatype(size), new column_name datatype(size)…); alter table table_name modify (column_name new datatype(new size),..);	alter table emp(add primary key (eno)); alter table student(add constraint city- chk check city in (‘pune’,‘mumbai’)); alter table student modify (city varchar(100));

The insert / update / delete statements

Name	Description	Syntax	Example
Insert	The insert statement is used to insert tuples or records into a created table or a relation.	insert into table_name values (list of column values);	insert into emp values(1,'joshi',4000,'pune);
	We specify a list of comma-separated column values, which must be in the same order as the columns in the table. To insert character data we must enclose it in single quotes('). If a single quote is part of the string value to be inserted , then precede it with a backslash(\).	insert into table_name(list of column names) values (list of column values corresponding to the column names);	insert into emp(eno,city) values(2,'mumbai');
	When we don't have values for every column in the table, or the data given in insert is not in the same column order as in the table, then we specify the column names also along with the values (2nd syntax)		
Update	The UPDATE command is used to change or modify data values in a table. To specify update of several columns at the same time, we simply specify them as a comma- separated list	update table_name set column_name = value where condition;	update emp set sal = sal +0.5*sal; update emp set sal = sal+1000 where eno =1;
Delete	The DELETE statement is used to remove data from tables.	delete from table_name where condition;	delete from emp ; delete from emp where eno = 1;

PostgreSQL provides many actions that allow you to:

- Add a column, drop a column, rename a column, or change a column's data type.
- Set a default value for the column.
- Add a CHECK constraint to a column.
- Rename a table.

The following illustrates the ALTER TABLE statement variants.

1. To **add a new column to a table** -

alter table add column statement:

ALTER TABLE table_name ADD COLUMN new_column_name TYPE;

2. To **remove an existing column-**

ALTER TABLE DROP COLUMN statement:

ALTER TABLE table_name DROP COLUMN column_name;

3. To **rename an existing column**

ALTER TABLE RENAME COLUMN TO statement:

ALTER TABLE table_name RENAME COLUMN column_name TO new_column_name;

4. To **change a default value** of the column

ALTER TABLE ALTER COLUMN SET DEFAULT or DROP DEFAULT:

ALTER TABLE table_name ALTER COLUMN [SET DEFAULT value | DROP DEFAULT]

5. To **change the NOT NULL constraint**

ALTER TABLE ALTER COLUMN statement:

ALTER TABLE table_name ALTER COLUMN [SET NOT NULL| DROP NOT NULL]

6. To **add a CHECK constraint**

ALTER TABLE ADD CHECK statement:

ALTER TABLE table_name ADD CHECK expression;

7. To **add a constraint**

ALTER TABLE ADD CONSTRAINT statement:

ALTER TABLE table_name ADD CONSTRAINT constraint_name constraint_definition

8. To **rename a table**

ALTER TABLE RENAME TO statement:

ALTER TABLE table_name RENAME TO new_table_name;

9. To **change the Data type of a column**

ALTER TABLE ALTER COLUMN column_name new_Data_type;

Set A

1) Create the table given below. Assume appropriate data types for attributes. Modify the table, as per the alter statements given below. Type \d <table name> and write the output.

Supplier (supplier_no int, supplier_name varchar (50), city char (10), phone_no int, amount float)

1. Make supplier_no as a primary key.
2. Add constraint city_check. the city should be in Pune, Mumbai, Nashik.
3. Drop the column phone_no.
4. Update the data type of supplier_name to text.
5. Remove the constraint city_check.
6. Remove the table Supplier.

2) Create the table given below. Assume appropriate data types for attributes. Modify the table, as per the alter statements given below. Type \d <table name> and write the output.

Student (stud_no int, stud_name varchar (50), address char (10), phone_no int, percentage float,class varchar(10))

1. Make stud_no as a primary key.
2. Add constraint per_check. The percentage should be greater than 35.
3. Change the column phone-no to mobile_no.
4. Update the data type of stud_name to text.
5. Rename the table to Student_Master.
6. Remove the column address from table.
7. Remove the table student.
8. Remove employee table from your database

Set B

1. Consider the following tables -

Supplier (supplier_no int, supplier_name varchar(50),city char(10),phone_no int, bill_amount float)

Constraint city_check.the city should be in Pune, Mumbai, Nashik and bill_amount greater than ZERO.

Student (stud_no int, stud_name varchar(50),address char(10),phone_no int ,percentage float,class varchar(10))

Constraint class_check.the class should be in FY, SY, TY.

Execute the queries for the following

1. Insert 5 records in Supplier table.
2. Insert 5 records in Student table.
3. Update Supplier table, increase the bill_amount by 5% percentage.
4. Update Supplier table, change the name of city from Nashik to Nagpur.

5. Update Student table increase the percentage by 3 whose class is FY.
6. Delete all Suppliers of Mumbai.
7. Delete students whose percentage is less than 40.

2. Create the following tables (primary keys are underlined.)
   Emp(eno, ename, designation, sal)
   Dept (dno, dname,dloc)

There exists a **one-to-many** relationship between Emp and Dept.

Create the Relations accordingly, so that the relationship is handled properly and the relations are in normalized form (3NF).

a. Insert 5 records into department table
b. Insert 2 employee records for each department.
c. Increase salary of "managers" by 15%;
d. Delete all employees of department 30;
e. Delete all employees who are working as a "clerk"
f. Change location of department Account to 'Nashik'.

Set C

1. Create the following tables (primary keys are underlined.)
Sales_order(s_order_no,s_order_date)

Client (client_no, name, address)

A client can give one or more Sales_orders, but a Sales_order belongs to exactly one client.

Create the Relations accordingly, so that the relationship is handled properly and the relations are in normalized form (3NF).

a. Insert 2 client records into client table
b. Insert 3 sales records for each client, change order date of client_no 'C004' to 12/6/2025
c. Delete all sale records having order date before 10/09/ 2024
d. Delete the record of client who lives in Mumbai.

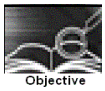
Assignment Evaluation

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

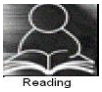
Practical In-charge

Assignment No. 4

Simple queries with select statement and aggregate function.



To understand and get a Hands-on on Select statement



You should read following topics before starting this exercise

1. Creating relations as tables and inserting tuples as records into the table.
2. The use of select statement in extracting information from the relation.
3. Insert/update/delete with subquery.
4. Relationship types & their conversion to RDB.
5. Normal forms 1NF, 2NF, 3NF.

The select statement:

Name	**Description**	**Syntax**	**Example**
Select statement	Used to read a tuple, tuples, parts of a tuple from a relation in the database. Tuple means a record in an RDB & a relation means a table. The basic structure of a select statement consists of 3 clauses: **The select clause** corresponds to the projection operation in relational algebra. It is used to list the attributes desired in the query. **The from clause** corresponds to the Cartesian product operation of RA. It lists the relations to be scanned in the evaluation of the expression. **The where clause** corresponds to the selection operation of RA. It consists of a predicate involving the attributes of the relations that appear in the select clause	select <attribute- list> from <relation- list> [where <condition> [group by <attribute list> [having <condition>] order by <attribute list>]];	select * from emp; Select eno,name from emp; select eno name from emp where sal > 4000 order by eno; select sum(sal) from emp group by dno having sum(sal)> 100000;

	The other clauses are:		
	Order by clause causes the result of the query to appear in a sorted order. **Group by clause** used to form groups of tuples, of the result. It is used when using aggregate functions. **Having clause** used with group by clause, to force a condition on the groups formed after applying group by clause, & selects only those groups in the output that satisfy the condition. The order of execution of the clauses is the same as given in the syntax.		

The aggregate functions supported by SQL are as follows:

Name	**Description**	**Usage**	**Example**
Sum()	Gets the sum or total of the values of the specified attribute.	Sum(attribute-name)	select sum(sal) from emp; select dno, sum(sal) from emp group by dno;
Count()	Gives the count of members in the group	Count(attribute); Count(*)	select count (*) from emp; select count(*) from emp where sal > 5000;
Max()	Gives the maximum value for an attribute, from a group of members	Max(attribute)	select max(sal) from emp; select dno, max(sal) from emp group by dno having count(*) >10;
Min()	Gives the minimum value for an attribute, from a group of members	Min(attribute)	select min(sal) from emp; select dno, min(sal) from emp group by dno having min(sal) >100;
Avg()	Gives the average value for an attribute, from a group of members	Avg(attribute)	select avg(sal) from emp; select dno, avg(sal) from emp group by dno having count(*) >10;

Set A

Create a table

Employee (empno, name, age, address, salary, deptno)

Insert atleast 10 records into the Employee table.

Execute following queries

1. Display all the details of employee.
2. Display distinct department numbers of employees.
3. Display the details of employee of deptno 5.
4. Display the details of employee who lives in Pune and having salary greater than 50000.
5. Display the details of employee whose age is between 25 and 35.
6. Display the employee number and employee name whose name start with S.
7. Display the details of employee whose name contain substring “nit”.
8. Display the maximum salary of employee.
9. Display average salary of employees.
10. Give the count of employees having age less than 35.
11. Display the total expenditure paid on employee salary.
12. Display the department wise total count of employees.

Consider Movie Database

Movies (M_name, release_year, budget)

Actor (A_name, role, charges, A_address, age)

Relationship: **Many to Many**

Create the relations accordingly, so that the relationship is handled properly and the relations are in normalized form(3NF).

Insert sufficient number of appropriate records.

Solve the Queries:

1. Display the names of actors whose name end with ‘n’.
2. List the names of the movies with the highest budget.
3. List the names of actors whose charges greater than 200000.
4. List the names of movies release in year 2023.
5. List the names of actors who has maximum charges.
6. List the names of actors who do not live in _____or _____ city.
7. List the names of movies whose budget is between 1cr to 5cr.

Set B

Consider the following database maintained by a school to record the details of all the competitions organized by the school every year. The school maintains information about competitions. Competition type can be like ‘academics’ or ‘sports’ etc.

Following are the tables:

Competition (C_no int, Name char (20), Type char (15), No_of_participants int, cdate date, budget float)

1. List out all the competitions held in the school.

2. List the names of all the competitions held in 2024.
3. Give the name of a competitions of type ‘Academic’.
4. Find out the total number of competitions organized in the school for competition type ‘sports ‘.
5. Find out maximum number of participants in competition.
6. Display average budget of competition of Sports.
7. Display competition name in which maximum number of participants are participated.
8. Display name of competition with minimum budget.
9. Display the total number of competition for each type.
10. Display the total budget of competition for each type.

Set C

Create the following tables (Primary keys are underlined):

Sailors(sid, sname, rate, age) Boats(bid, bname, colour) Reserves(sid, bid, date)

Sailors and boats are related **many to many**.

Create the relations accordingly, so that the relationship is handled properly and the relations are in normalized form(3NF) and insert 5 records into each table.

Draw ER diagram for given relational schema and show normalization. Solve the following queries:

a) Find all the sailors with a rating above 8.

b) Find the ages of sailors whose name begins and ends with ‘P’.

c) Find name of sailors who have reserved red and green boats.

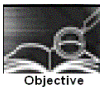
Assignment Evaluation

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

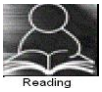
Practical In-charge

Assignment No.5

Queries with set operations



To understand & get a Hands-on on using set operations (union, intersect and except) with select statement.



You should read following topics before starting this exercise

1. Relation algebra operation
2. SQL operations union, intersect & except
3.

SQL Set operations:

Name	**description**	**Syntax**	**Example**
Union	Returns the union of two sets of values, eliminating duplicates.	<select query> union <select query>	select cname from depositor union select cname from borrower;
Union all	Returns the union of two sets of values, retaining all duplicates.	<select query> Union all <select query>	select cname from depositor union all select cname from borrower;
Intersect	Returns the intersection of two sets of values, eliminating duplicates	<select query> intersect <select query>	select cname from depositor intersect select cname from borrower;
Intersect all	Returns the intersection of two sets of values, retaining duplicates	<select query> Intersect all <select query>	select cname from depositor intersect all Select cname from borrower;
except	Returns the difference between two set of values, I.e returns all values from set1, not contained in set2 .eliminates duplicates	<select query> except <select query>	select cname from depositor except select cname from borrower;
Except all	Returns the difference between two set of values, i.e. returns all values from set1, not contained in set2 Retains all duplicates	<select query> Except all <select query>	select cname from depositor except all select cname from borrower;

Note: To use the INTERSECT operator, the columns that appear in the SELECT statements must follow the rules below:

1. The number of columns and their order in the SELECT clauses must be the same.
2. The data types of the columns must be compatible.

Set A

Create the following relations, for an investment firm

emp(emp_id ,emp_name, address, bdate)

Investor (inv_no, inv_name, inv_date, inv_amt)

An employee may invest in one or more investments; hence he can be an investor. But an investor need not be an employee of the firm.

Create the Relations accordingly, so that the relationship is handled properly and the relations are in normalized form (3NF).

Assume appropriate data types for the attributes. Add any new attributes, as required by the queries. Insert sufficient number of records in the relations / tables with appropriate values as suggested by some of the queries**.**

Write the following queries & execute them.

1. List the distinct names of customers who are either employees, or investors or both.
2. List the names of customers who are either employees, or investors or both.
3. List the names of employees who are also investors.
4. List the names of employees who are not investors

Set B

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

Project (project_no, project_name, status)

Department (Dept_no, dept_name, location)

Constraints: Employee designation is either 'manager', 'staff', 'worker'.

There exists a **one-to-many** relationship between Department and Employee. Many employees can work on many projects controlled by a department. Create the relations accordingly, so that the relationship is handled properly and the relations are in normalized form (3NF) and insert 5 records into each table.

Solve the following queries:

1. Find the details of employee who is having highest salary.
2. Delete all employees of department 20.
3. List the names and salary of employees sorted by their salary.

Assignment Evaluation

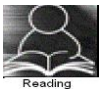
0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment No.6

Nested Queries and Views

To understand & get a Hands-on on nested queries & subqueries, that involves joining of tables, to demonstrate set cardinality.



You should read following topics before starting this exercise

1. Nesting of SQL queries and subqueries
2. SQL statements involving set membership, set comparisons and set cardinality operations.

SQL includes a feature for testing whether a subquery has any tuples in its result, using the following clauses:

A sub-query is a select-from-where expression that is nested within another query.

Set membership	the 'in' & 'not in' connectivity tests for set membership & absence of set membership respectively.
Set comparison	the < some, > some, <= some, >= some, = some, <> some are the constructs allowed for comparison. = some is same as the 'in' connectivity. <> some is not the same as the 'not n'i connectivity. Similarly, sql also provides < all, >all, <=all, >= all, <> all comparisons. <>all is same as the 'not in' construct.
Set cardinality	The 'exists' construct returns the value true if the argument subquery is nonempty. We can test for the non-existence of tuples in a subquery by using the 'not exists' construct. The 'not exists 'construct can also be used to simulate the set containment operation (the super set). We can write "relation A contains relation B" as "not exists (B except A)".

Views

Views can be thought of as stored queries, which allow us to create a database object that functions very similarly to a table, but whose contents dynamically reflect the selected rows. Views are very flexible; you may use a view to address common simple queries to a single table, as well as for complicated ones which may span across several tables.

Creating a View

CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW name [(column_name [, ...])] AS query CREATE VIEW defines a view of a query. The view has no physical existence, but is dynamically created whenever it is referenced in a query.

CREATE OR REPLACE VIEW replaces an existing view of same name. The new query must use the same column names in the same order and with the same data types, but it may add additional columns to the end of the list.

Schema name (CREATE VIEW myschema.myview) must be specified to create a view in schema other than the current schema. The TEMPORARY or TEMP parameter is specified to create temporary views, however if any of the tables referenced by the view are temporary, the view is created as a temporary view (whether TEMPORARY is specified or not). Temporary views exist in a special schema, so a schema name cannot be given when creating a temporary view. View names should be distinct.

Name: Name (optionally schema-qualified) of a view to be created.

column_name: An optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.

Query: A SELECT command which will provide the columns and rows of the view.

Example to create a view consisting of all comedy films

```
CREATE VIEW movie AS
        SELECT *
        FROM films
        WHERE type = 'bollywood';
```

ALTER VIEW statement is used to change the definition of a view.

ALTER VIEW name ALTER [COLUMN] column SET DEFAULT expression

ALTER VIEW name ALTER [COLUMN] column DROP DEFAULT

DROP VIEW statement is used to remove a view

DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]

Set A

1. Create the following relations:

Emp(eno, name,dno,salary)

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

Each employee can work on one or more projects, and a project can have many employees working in it. The number of hours worked on each project, by an employee also needs to be stored. Create the Relations accordingly, so that the relationship is handled properly and the relations are in normalized form (3NF).

Assume appropriate data types for the attributes. Add any new attributes, new relations as required by the queries. Insert sufficient number of records in the relations / tables with appropriate values as suggested by some of the queries.

a. Write the queries for following business tasks & execute them.

1. List the names of departments that controls project whose budget is greater than ___.
2. List the names of projects, controlled by department No __, whose budget is greater than atleast one project controlled by department No __.
3. List the details of the projects with second maximum budget.
5. List the names of employees, working on some projects that employee number __ is working.
6. List the names of employees who do not work on any project that employee number __ works on
7. List the names of employees who do not work on any project controlled by '_____'department

8. List the names of projects along with the controlling department name, for those projects which has atleast __ employees working on it.
9. List the names of employees who is worked for more than 10 hrs. on atleast one project controlled by '_____' dept.
10. List the names of employees, who are males, and earning the maximum salary in their department.
11. List the names of employees who work in the same department as '_____'.
12. List the names of employees who do not live in _____ or _____.

b. Create view for the following:

1. Display all employees working on "ERP" Project.
2. Display the project details and start date of the project, sort it by start date of the project where duration of project is more than 6 months.
3. Display employee details having qualification MCA.
4. Display employee and project names where employees worked more than 300 hours.
5. Delete view where employee having qualification MCA.

Set B

1. 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 (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).

Solve the Queries

1. Find the names of the customers for the "Aundh" branch.
2. List the names of the customers who have received loan less than their requirement.
3. Find the maximum loan amount approved.
4. Find out the total loan amount sanctioned by "Deccan "branch.
5. Count the number of loan applications received by "M.G. ROAD" branch.
6. List the names of the customer along with the branch names who have applied for loan in the month of September.

2. Create view for the following:

a. Display the details of all customers who have received loan amount less than their requirement.

b. Display sum of loan amount approved branch wise from 1st June 2019 to 1st June 2020.

2. Count branch wise all customers who required loan amount more than 30 lakhs.

3. Display all customer names branch wise who requested loan amount less than one lakh.

3. Employee-Project database

Create the following relations:

Emp (eno, name, dno,salary)

Project (pno, pname,control_dno,budget)

Each employee can work on one or more projects, and a project can have many employees working in it. The number of hours worked on each project, by an employee also needs to be stored.

Create the Relations accordingly, so that the relationship is handled properly and the relations are in normalized form (3NF).

Assume appropriate data types for the attributes. Add any new attributes, new relations as required by the queries.

Insert sufficient number of records in the relations / tables with appropriate values as suggested by some of the queries**.**

1. List the names of employees who work in all the projects that "____" works on.
2. List the names of employees who work on only some projects that "." works on
3. List the names of the departments that have atleast one project under them. (write using 'exists 'clause)
4. List the names of employees who do not work on "sales" project (write using 'not exists') clause
5. List the names of employees who work only on those projects that are controlled by their department.
6. List the names of employees who do not work on any projects that are controlled by their department

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: **M-M** with descriptive attribute Subject.

Solve the queries

1. Find the minimum experienced teacher.
2. Find the number of teachers having qualification “Ph. D.”.
3. List the names of the students to whom “Mr. Patil” is teaching along with the subjects he is teaching to them.
4. Find the subjects taught by each teacher.
5. List the names of the teachers who are teaching to a student named “Suresh”.
6. List the names of all teachers along with the total number of students they are teaching.

Assignment Evaluation

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

Practical In-charge

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)

Field Project using Software Engineering Techniques

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

Major: Computer Science

CS-231-FP Semester III

Field Project

Name: _____

College Name: _____

Roll No.: _____ Division: _____

Academic Year: _____

Submission Deadline: / /

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

Academic Year (20 - 20)

Project Title:_____

Team members:

1) **Name :** _____

Roll No . **Exam Seat No:**

2) **Name :** _____

Roll No . **Exam Seat No:**

Project Guide Name: _____

Project Guide Signature: _____

Start Date: / / Completion Date: / /

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that **Ms./Mr. _____** have successfully and satisfactorily completed and submitted the field project titled_____ in partial fulfilment of **S.Y.B.Sc.(CS)** Lab course **CS-231-FP Semester III** prescribed by Savitribai Phule Pune University during the academic year _____.

(Project Guide) (H.O.D. Computer Science)

INTERNAL EXAMINER EXTERNAL EXAMINER

Savitribai Phule Pune University
S.Y.B.Sc. (Computer Science) - Sem – III
Course Type: FP/OJT/CEP
Course Code: CS-231-FP
Course Title : Field Project

Teaching Scheme	No. of Credits	Examination Scheme
4 Hours/Week	**2**	IE: 15 marks UE: 35 marks

Prerequisites

ER Modeling

Course Objectives

1. To get knowledge and understanding of software engineering discipline.
2. To learn analysis and design principles for software project development.
3. To Implement Agile Development Methodologies in real life Software Projects.

Course Outcomes

On completion of the course, student will be able to-

CO1: Identify requirements, analyze and prepare models.

CO2: Understand basic Software engineering concepts and Process models.

CO3: Choose a process model for a software project development.

CO4: Design different UML Diagrams.

Course Contents

Assignment No	Title	No of hours
1	Preliminary Investigation and its activities	12 Hours
2	Requirement Specification	12 Hours
3	Database Design	12 Hours
4	System Design	12 Hours

Index