AMAL JYOTHI COLLEGE OF ENGINEERING (AUTONOMOUS)

KANJIRAPALLY

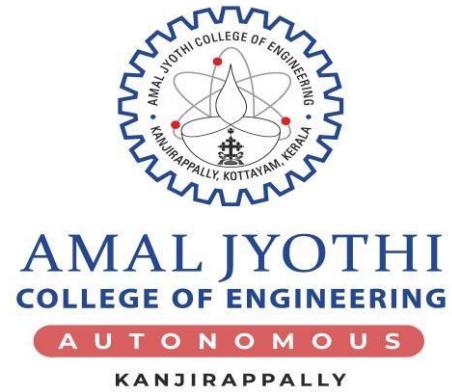
CSL333 - DATABASE MANAGEMENT SYSTEMS LAB RECORD

Department of Computer Science & Engineering

November – 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMAL JYOTHI COLLEGE OF ENGINEERING, KANJIRAPPALLY



Name : …….……

Semester : …… Branch : ……..……..

Batch : ……. Roll No. : ……..……..……..

University Reg No. : ……...……

Certified that this is a bonafide record of practical work done in **CSL 333 – DATABASE MANAGEMENT SYSTEMS** *Laboratory by*

…… *during the year* **2024-2025.**

Head of the Department

Faculty in Charge

Internal Examiner

External Examiner

Vision of the Department

The Computer Science and Engineering department is committed to continually improve the educational environment in order to develop professionals with strong technical and research backgrounds.

Mission of the Department

To provide quality education in both theoretical and applied foundations of Computer Science & Engineering.
Create highly skilled Computer Engineers, capable of doing research and also develop solutions for the betterment of the nation.
Inculcate professional and ethical values among students.
Support society by participating in and encouraging technology transfer.

Programme Educational Objectives

PE01 Be successfully employed in computing profession as well as multidisciplinary domains in supportive and leadership roles.
PE02 Participate in life-long learning through successful completion of advanced degrees, continuing education, certifications and/or other professional development.
PE03 Promote design, research, product implementation and services in the field of Computer Science and Engineering through strong technical, communication and entrepreneurial skills.

Programme Outcomes

B Tech Computer Science and Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes

PS01 Apply Engineering knowledge to analyze, design and develop computing solutions by employing modern computer languages, environments and platforms that can solve complex problems.
PS02 Anticipate the changing direction of computational technology, evaluate it and communicate the likely utility of that for building software systems that would perform tasks related to industry, research and education.
PS03 Inculcate the knowledge of Engineering and Management principles to manage projects effectively and create innovative career paths.

CONTENTS

Date:_____

EXPERIMENT -1
SQL FAMILIARIZATION

AIM

Draw an ER Diagram and Create a table student with the following fields:sid, sname, dept, mark1, mark2.Insert 5 values to the table and display.

Then

a. Add grade column to the student.

b. Rename grade column to CGPA.

c. Modify data type of sname of student table from varchar to char.

d. Drop column mark2 from student table.

e. Rename student table to student details.

f. Show the structure of table student

COURSE OUTCOME

CO1: Design database schema for a given real world problem-domain using standard design and modeling approaches.
CO2: Construct queries using SQL for database creation, interaction, modification, and updation.

TABLE CREATION

// Create a table student with the following fields.sid, sname, dept, mark1, mark2.

Query- CREATE TABLE Student (SID int primary key, SNAME varchar(20), DEPT varchar(20), MARK1 int,MARK2 int);

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SID                                       NOT NULL NUMBER(38)
SNAME                                              VARCHAR2(20)
DEPT                                               VARCHAR2(20)
MARK1                                              NUMBER(38)
MARK2                                              NUMBER(38)
```

// **Insert 5 values to the table and display.**
Query- insert into student values(&sid,&sname,&dept,&mark1,&mark2);
Enter value for sid: 1
Enter value for sname: 'Abhishek'
Enter value for dept: ‘CSE'
Enter value for mark1: 45
Enter value for mark2: 40

Enter value for sid: 2
Enter value for sname: 'Ashique'
Enter value for dept: ‘CSE'
Enter value for mark1: 47
Enter value for mark2: 50

Enter value for sid: 3
Enter value for sname: 'Siby'
Enter value for dept: ‘CSE'
Enter value for mark1: 50
Enter value for mark2: 50

Enter value for sid: 4
Enter value for sname: 'Advaith'
Enter value for dept: ‘CSE'
Enter value for mark1: 40
Enter value for mark2: 50

Enter value for sid: 5
Enter value for sname: 'Shibu'
Enter value for dept: ‘CSE'
Enter value for mark1: 25
Enter value for mark2: 30

SID	SNAME	DEPT	MARK1	MARK2
1	Abhishek	CSE	45	40
2	Ashique	CSE	47	50
3	Siby	CSE	50	50
4	Advaith	CSE	40	50
5	Shibu	CSE	25	30

QUERIES

a. Add grade column to the student.

Query

1. alter table add grade varchar(5);

```
Table altered.
```

2.desc student;

Name	Null?	Type
SID	NOT NULL	NUMBER(38)
SNAME		VARCHAR2(20)
DEPT		VARCHAR2(20)
MARK1		NUMBER(38)
MARK2		NUMBER(38)
GRADE		VARCHAR2(5)

b. Rename grade column to CGPA.

Query

1.alter table student rename column grade to CGPA;

```
Table altered.
```

2.desc student;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SID                                       NOT NULL NUMBER(38)
SNAME                                              VARCHAR2(20)
DEPT                                               VARCHAR2(20)
MARK1                                              NUMBER(38)
MARK2                                              NUMBER(38)
CGPA                                               VARCHAR2(5)
```

c. Modify data type of sname of student table from varchar to char.
Query

1.alter table student modify sname char(20);

```
Table altered.
```

2.desc student;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SID                                       NOT NULL NUMBER(38)
SNAME                                              CHAR(20)
DEPT                                               VARCHAR2(20)
MARK1                                              NUMBER(38)
MARK2                                              NUMBER(38)
CGPA                                               VARCHAR2(5)
```

d. Drop column mark2 from student table.
Query-
1.alter table student drop column marks2

```
Table altered.
```

2.desc student;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SID                                       NOT NULL NUMBER(38)
SNAME                                              CHAR(20)
DEPT                                               VARCHAR2(20)
MARK1                                              NUMBER(38)
CGPA                                               VARCHAR2(5)
```

e. Rename student table to student details.

Query-

1. alter table student rename to studentdetails

```
Table altered.
```

2.desc studentdetails;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SID                                       NOT NULL NUMBER(38)
SNAME                                              CHAR(20)
DEPT                                               VARCHAR2(20)
MARK1                                              NUMBER(38)
CGPA                                               VARCHAR2(5)
```

f. Show the structure of table student.

Query-

1. Desc studentdetails;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SID                                       NOT NULL NUMBER(38)
SNAME                                              VARCHAR2(20)
DEPT                                               VARCHAR2(20)
MARK1                                              NUMBER(38)
MARK2                                              NUMBER(38)
```

RESULT

SQL queries are executed and output was obtained, thus C01 and CO2 achieved .

Date:_____

EXPERIMENT -2
SQL FAMILIARIZATION-EMPLOYEE

AIM

Create a table employee. The attributes of employee are empno, empname, place, desig, salary, dept_no, dept_name.Set empno as primary key and also set 10000 as a default salary of employee.Create a constraint where the dept_no ranges from 1 -10.
a. Display the empno, empname and salary.
b. Display all records of employee table
c. Display the details of the employee whose salary is greater than 10000.
d. Increment the salary of 'officer' by 1000 .
e. Display the name and id of employee whose salary ranges between 1.
10000 and 15000.
f. Display the employee details whose name start with 'A'
g. Change the salary of officer to 15000
h. Add a column district to the table
i. Delete the column district
j. Modify the column name desig to designation

COURSE OUTCOME

CO1: Design database schema for a given real world problem-domain using standard design and modeling approaches.
CO2: Construct queries using SQL for database creation, interaction, modification, and updation.

TABLE CREATION

//Create a table employee. The attributes of employee are empno, empname, place, desig, salary, dept_no, dept_name.Set empno as primary key and also set 10000 as a default salary of employee.Create a constraint where the dept_no ranges from 1 -10

Query - create table employee(empno int primary key,empname varchar(25),place varchar(10),desig varchar(25),salary int default 10000,deptno int check(deptno between 1 and 10),deptname varchar(25));

```
SQL> desc employee;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 EMPNO                                     NOT NULL NUMBER(38)
 EMPNAME                                            VARCHAR2(25)
 PLACE                                              VARCHAR2(10)
 DESIG                                              VARCHAR2(25)
 SALARY                                             NUMBER(38)
 DEPTNO                                             NUMBER(38)
 DEPTNAME                                           VARCHAR2(25)
```

//**INSERTING VALUES**
EMPLOYEE TABLE
QUERY- insert into employee values
(&empno,&empname,&place,&desig,&salary,&deptno,&deptname);
Enter value for empno: 1
Enter value for empname: 'Thomas'
Enter value for place: ‘Kollam'
Enter value for desig: 'officer'
Enter value for salary: '12000'
Enter value for deptno: '1'
Enter value for deptname: 'design'
Enter value for empno: 2
Enter value for empname: 'Avin'
Enter value for place: ‘Kottayam'
Enter value for desig: 'hod'
Enter value for salary: '20000'
Enter value for deptno: '2'
Enter value for deptname: 'marketing'
Enter value for empno: 3
Enter value for empname: 'Arjun'
Enter value for place: ‘Idukki'
Enter value for desig: 'Advisor'
Enter value for salary: '30000'
Enter value for deptno: '3'
Enter value for deptname: 'management

QUERIES

a. Display the empno, empname and salary.
Query- select empno,empname,salary from employee;

EMPNO	EMPNAME	SALARY
1	Thomas	12000
2	avin	20000
3	Arjun	30000
4	Farhan	12000

b. Display all records of employee table.
Query- select * from employee;

EMPNO	EMPNAME	PLACE	DESIG	SALARY	DEPTNO	DEPTNAME
1	Thomas	kollam	officer	12000	1	design
2	avin	kottayam	hod	20000	2	marketing
3	Arjun	Idukki	Advisor	30000	3	management
4	Farhan	Kannur	salesperson	12000	1	design

c. Display the details of the employee whose salary is greater than 10000.
Query- select * from employee where salary >10000;

EMPNO	EMPNAME	PLACE	DESIG	SALARY	DEPTNO	DEPTNAME
1	Thomas	kollam	officer	12000	1	design
2	avin	kottayam	hod	20000	2	marketing
3	Arjun	Idukki	Advisor	30000	3	management
4	Farhan	Kannur	salesperson	12000	1	design

d. Increment the salary of ‘officer’ by 1000
Query-
1.update employee set salary=salary+1000 where desig=’officer’;

```
1 row updated.
```

2.Select * from employee where desig = ‘officer;

EMPNO	EMPNAME	PLACE	DESIG	SALARY	DEPTNO	DEPTNAME
1	Thomas	kollam	officer	13000	1	design

h. Add a column district to the table
Query-
1.alter table employee add district varchar(10);

```
Table altered.
```

2.desc employee;

```
Name                                              Null?    Type
------------------------------------------------- -------- ---------------
EMPNO                                             NOT NULL NUMBER(38)
EMPNAME                                                    VARCHAR2(25)
PLACE                                                      VARCHAR2(10)
DESIG                                                      VARCHAR2(25)
SALARY                                                     NUMBER(38)
DEPTNO                                                     NUMBER(38)
DEPTNAME                                                   VARCHAR2(25)
DISTRICT                                                   VARCHAR2(10)
```

i. Delete the column district

Query-
1.alter table employee drop column district;

```
Table altered.
```

2.desc employee;

```
Name                                              Null?    Type
------------------------------------------------- -------- ------------
EMPNO                                             NOT NULL NUMBER(38)
EMPNAME                                                    VARCHAR2(25)
PLACE                                                      VARCHAR2(10)
DESIG                                                      VARCHAR2(25)
SALARY                                                     NUMBER(38)
DEPTNO                                                     NUMBER(38)
DEPTNAME                                                   VARCHAR2(25)
```

j. Modify the column name desig to designation
Query-
1.alter table employee rename column desig to designation;

```
Table altered.
```

2.desc employee;

```
Name                                              Null?    Type
------------------------------------------------- -------- ------------
EMPNO                                             NOT NULL NUMBER(38)
EMPNAME                                                    VARCHAR2(25)
PLACE                                                      VARCHAR2(10)
DESIGNATION                                                VARCHAR2(25)
SALARY                                                     NUMBER(38)
DEPTNO                                                     NUMBER(38)
DEPTNAME                                                   VARCHAR2(25)
```

RESULT

SQL queries are executed and output was obtained, thus C01 and CO2 achieved .

Date:_____

EXPERIMENT -3
SQL FAMILIARIZATION-CUSTOMER

AIM

Create a table customer with the following fields: customerid, name, branch, accno, balance. Customerid is the primary key. In all other fields, we cannot enter null value. The balance should not be less than 500.
a. Find out the details of all customers whose balance is between 2000 and 3000.
b. Show all branches of the bank (duplicates eliminated).
c. Find out the details of all customers whose branch is kottayam and balance>5000.
d. Show the details of all customers whose name start with A.
e. Retrieve the branch name values as city.
f. Find the total balance of the bank.
g. Find the average balance of the bank.
h. Find the max value for balance.
i. Find the min balance of the bank.
j. Count number of records in the table.
k. Modify the size of name in the table to 50
l. Add a new column address t the table with data type varchar(10) and insert values into it.

COURSE OUTCOME

CO1: Design database schema for a given real world problem-domain using standard design and modeling approaches.
CO2: Construct queries using SQL for database creation, interaction, modification, and updation.

TABLE CREATION

//Create a table customer with the following fields: customerid, name, branch, accno, balance. Customerid is the primary key. In all other fields, we cannot enter null value. The balance should not be less than 500.

Query-create table customer(customerid int primary key,name varchar(20) not null,branch varchar(20) not null,accno int not null,balance int not null check(balance>500));

```
Name                                              Null?    Type
------------------------------------------------- -------- ------------
CUSTOMERID                                        NOT NULL NUMBER(38)
NAME                                              NOT NULL VARCHAR2(20)
BRANCH                                            NOT NULL VARCHAR2(20)
ACCNO                                             NOT NULL NUMBER(38)
BALANCE                                           NOT NULL NUMBER(38)
```

//INSERTING VALUES
CUSTOMER TABLE
Query- insert into customer values(&customerid,&name,&branch,&accno,&balance);
Enter value for customerid: 100
Enter value for name: 'Prince'
Enter value for city: ‘Kottayam'
Enter value for accno: '90996677'
Enter value for balance: '7000'

Enter value for customerid: 102
Enter value for name: 'Isac'
Enter value for city: ‘Kozhikode'
Enter value for accno: '90998833'
Enter value for balance: '4000'
Enter value for customerid: 104
Enter value for name:’Anandhu'
Enter value for city: ‘Kottayam'
Enter value for accno: '9008865'
Enter value for balance: '2600'

QUERIES

a. Find out the details of all customers whose balance is between 2000 and 3000.
Query- select * from customer where balance between 2000 and 3000;

CUSTOMERID	NAME	BRANCH	ACCNO	BALANCE
104	Anandhu	Kottayam	9008865	2600
106	Gokul	Ernakulam	9008861	2800

b. Show all branches of the bank (duplicates eliminated).
Query- select distinct branch from customer;

BRANCH
Kottayam
Kozhikode
Ernakulam

c. Find out the details of all customers whose branch is kottayam and balance>5000.
Query- select * from customer where branch=’Kottayam’ and balance>5000;

CUSTOMERID	NAME	BRANCH	ACCNO	BALANCE
100	Prince	Kottayam	90996677	7000

d. Show the details of all customers whose name start with A.
Query- select * from customer where name like ‘A%’;

CUSTOMERID	NAME	BRANCH	ACCNO	BALANCE
104	Anandhu	Kottayam	9008865	2600
103	Abhinay	Kozhikode	900997788	8000

e. Retrieve the branch name values as city.
Query- select branch as city from customer;

CITY
Kottayam
Kozhikode
Ernakulam

f. Find the total balance of the bank.
Query- select sum(balance) from customer;

SUM(BALANCE)
24400

g. Find the average balance of the bank.
Query- select avg(balance) from customer;

AVG(BALANCE)
4880

h. Find the max value for balance.
Query-select max(balance) from customer;

MAX(BALANCE)
8000

i. Find the min balance of the bank.
Query- select min(balance) from customer;

MIN(BALANCE)
2600

j. Count number of records in the table.
Query- select count(*) from customer;

COUNT(*)
5

k. Modify the size of name in the table to 50
Query-

1. alter table customer modify name varchar(50);

```
Table altered.
```

2. desc customer;

```
Name                                                  Null?    Type
----------------------------------------------------- -------- ---------------
CUSTOMERID                                            NOT NULL NUMBER(38)
NAME                                                  NOT NULL VARCHAR2(50)
CITY                                                  NOT NULL VARCHAR2(20)
ACCNO                                                 NOT NULL NUMBER(38)
BALANCE                                               NOT NULL NUMBER(38)
```

l. Add a new column address to the table with data type varchar(10) and insert values into it.
Query-
1. alter table customer add address varchar(10);

```
Table altered.
```

2. desc customer;

```
Name                                                  Null?    Type
----------------------------------------------------- -------- -------------
CUSTOMERID                                            NOT NULL NUMBER(38)
NAME                                                  NOT NULL VARCHAR2(50)
CITY                                                  NOT NULL VARCHAR2(20)
ACCNO                                                 NOT NULL NUMBER(38)
BALANCE                                               NOT NULL NUMBER(38)
ADDRESS                                                        VARCHAR2(10)
```

3. update customer set address=&address where name =&name;
Enter value for address: ‘pampadi’
Enter value for name: ‘Prince’
Enter value for address: ‘Kozhikode’
Enter value for name: ‘Isac’

```
CUSTOMERID NAME                                               CITY                    ACCNO    BALANCE ADDRESS
---------- -------------------------------------------------- -------------------- ---------- ---------- ----------
       100 Prince                                             Kottayam             90996677       7000 pampadi
       102 Isac                                               Kozhikode            90998833       4000 kozhikode
       104 Anandhu                                            Kottayam              9008865       2600 erumely
       103 Abhinay                                            Kozhikode           900997788       8000 kozhikode
       106 Gokul                                              Ernakulam             9008861       2800 vypin
```

RESULT

SQL queries are executed and output was obtained, thus CO1 and CO2 achieved.

Date:_____

EXPERIMENT-4
SQL FAMILIARIZATION-STUDENT

AIM

Create a table student with following fields roll no int (primary key), Name char (20) not null (first letter as either B,S E,P), sex char (1) accept only m or f, dob date not null, course (values must be MCA, CSE ME), sem(values must be S3, S4), Date_of_Join.

Create second table marks with following data Mid in (primary key), roll no int (foreign key) referencing student tables).
Sub_code char (5) not null and marks int not null (>=0
&<=100). Insert the data into these tables.

a. List the name of students joined in mca after 10-10-1990.
b. List the name of students who are not in CS department.
c. List the names of students whose names start with 'E' and 'P as 3rd character.
d. List all marks of the student Sourav from MCA.
e. List all roll no from two table (avoid duplicate roll no).
f. List all roll no which is common in both tables.
g. List name from student table and all marks from marks of roll no 23 in student table.
h. List the roll no and total marks of each roll no from mark table.
i. Display name and roll no of students, where marks are entered in marks table.
j. Display the name, roll no, sex, dob, sub_code and mark of highest subject mark.
k. List the student name and Date of Join in format dd/mm/yy
l. List all students joined during the year 1998
m. List the minimum mark of various students in various department having minimum mark greater than 60.
n. List all the students in the college other than CS Department
o. Count the number of students in each department whose mark in greater than 60.

COURSE OUTCOME

CO1: Design database schema for a given real world problem-domain using standard design and modeling approaches.

CO2: Construct queries using SQL for database creation, interaction, modification, and updation.

TABLE CREATION

// Create a table student with following fields roll no int (primary key), Name char (20) not null (first letter as either B,S E,P), sex char (1) accept only m or f, dob date not null, course (values must be MCA, CSE ME), sem(values must be S3, S4), Date_of_Join.

Query- create table student(rollno int primary key, Name char (20) not null check(name like 'B%' or name like 'S%' or name like 'E%' or name like 'P%'), sex char (1) check(sex in('f','m')),dob date not null, course varchar(5) check(course in('MCA','CSE','ME')),sem varchar(5) check(sem in ('S3','S4')), Date_of_Join date not null);

```
SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ROLLNO                                    NOT NULL NUMBER(38)
 NAME                                      NOT NULL CHAR(20)
 SEX                                                CHAR(1)
 DOB                                       NOT NULL DATE
 COURSE                                             VARCHAR2(5)
 SEM                                                VARCHAR2(5)
 DATE_OF_JOIN                              NOT NULL DATE
```

Create second table marks with following data Mid in (primary key), roll no int (foreign key) referencing student tables). Sub_code char (5) not null and marks int not null (>=0 &<=100). Insert the data into these tables.

Query – create table marks(mid int primary key,rollno int references student(rollno),Sub_code char(5) not null,marks int not null check(marks>=0 and marks<=100));

```
SQL> desc marks;
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------
 MID                                       NOT NULL NUMBER(38)
 ROLLNO                                             NUMBER(38)
 SUB_CODE                                  NOT NULL CHAR(5)
 MARKS                                     NOT NULL NUMBER(38)
```

//INSERTING VALUES
STUDENT TABLE
QUERY-
insert into student values(&rollno,&name,&sex,&dob,&course,&sem,&date_of_join);
Enter value for rollno: 1
Enter value for name:'Babu'
Enter value for sex:'m'
Enter value for dob: '12-oct-1975'
Enter value for course: 'CSE'
Enter value for sem: 'S3'
Enter value for date_of_join: '13-jan-1989'
Enter value for rollno: 2
Enter value for name: 'Praveen'
Enter value for sex: 'm'
Enter value for dob: '20-feb-1967'
Enter value for course: 'MCA'

Enter value for sem: 'S4'
Enter value for date_of_join: '15-aug-1998'
Enter value for rollno: 3
Enter value for name: 'Empoy'
Enter value for sex: 'f'
Enter value for dob: '18-jul-1977'
Enter value for course: 'ME'
Enter value for sem: 'S3'
Enter value for date_of_join: '20-mar-2000'

MARKS TABLE
QUERY-
insert into marks values(&mid,&rollno,&sub_code,&marks);
Enter value for mid: 100
Enter value for rollno: 23
Enter value for marks: 95
Enter value for mid: 101
Enter value for rollno: 2
Enter value for sub_code:'JAV09'
Enter value for marks: 88
Enter value for mid: 102
Enter value for rollno: 3
Enter value for sub_code: 'MPC80'
Enter value for marks: 90

QUERIES

a. List the name of students joined in mca after 10-10-1990.
Query-
select name from student where course='MCA' and date_of_join>'10- oct-1990';

```
NAME
--------------------
Praveen
Sourav
```

b. List the name of students who are not in CS department.
Query-
select name from student where course !='CSE';

```
NAME
--------------------
Praveen
Empoy
Sourav
```

c. List the names of students whose names start with 'E' and 'P as 3rd character
Query-
select * from student where name like 'E_p%';

```
SQL> select * from student where name like 'E_p%';

    ROLLNO NAME                 S DOB       COURS SEM   DATE_OF_J
---------- -------------------- - --------- ----- ----- ---------
         3 Empoy                f 18-JUL-77 ME    S3    20-MAR-00
```

d. List all marks of the student Sourav from MCA.
Query-
select s.name,m.sub_code,m.marks from student s,marks m where s.rollno=m.rollno and name='Sourav';

```
NAME                 SUB_C      MARKS
-------------------- ----- ----------
Sourav               DSP07         93
Sourav               CNW01         78
```

e. List all roll no from two table (avoid duplicate roll no).
Query-
select rollno from student union select rollno from marks;

```
    ROLLNO
----------
         1
         2
         3
         4
        23
```

f. List all roll no which is common in both tables.
Query-
select rollno from student intersect select rollno from marks;

```
    ROLLNO
----------
         1
         2
         3
         4
```

g. List name from student table and all marks from marks of roll no 23 in student table.
Query-
select s.name,m.marks from student s,marks m where s.rollno=m.rollno and s.rollno=23;

```
NAME                      MARKS
-------------------- ----------
Bibin                        95
Bibin                        67
```

h. the roll no and total marks of each roll no from mark table.
Query-

select rollno,sum(marks) as TOTAL_MARKS from marks group by rollno;

ROLLNO	TOTAL_MARKS
2	163
4	171
23	162
3	155

i. Display name and roll no of students, where marks are entered in marks table.
Query-
select s.name,m.marks from student s,marks m where s.rollno=m.rollno;

NAME	MARKS
Bibin	95
Praveen	88
Empoy	90
Sourav	93
Sourav	78
Empoy	65
Praveen	75
Bibin	67

j. Display the name, roll no, sex, dob, sub_code and mark of highest subject mark.
Query- select s.name,s.rollno,s.sex,s.dob,m.sub_code,marks from student s,marks m wheres.rollno=m.rollno and (marks,s.rollno) in(select max(marks),rollno from marks group by rollno);

NAME	ROLLNO	S	DOB	SUB_C	MARKS
Bibin	23	m	24-OCT-67	MAT10	95
Empoy	3	f	18-JUL-77	MPC80	90
Praveen	2	m	20-FEB-67	JAV09	88
Sourav	4	m	25-APR-88	DSP07	93

k. List the student name and Date of Join in format dd/mm/yy
Query- select name,to_char(date_of_join,'dd/mm/yy') from student;

NAME	TO_CHAR(
Babu	13/01/89
Praveen	15/08/98
Empoy	20/03/00
Sourav	07/01/98
Bibin	25/02/89

l. all students joined during the year 1998
Query- select name from student where extract(year from date_of_join)=1998;

NAME
Praveen
Sourav

m. List the minimum mark of various students in various department having minimum mark greater than 60.
Query- select s.name,s.course,min(marks) from student s,marks m where s.rollno=m.rollno group by(name,course) having min(marks)>60;

NAME	COURS	MIN(MARKS)
Sourav	MCA	78
Praveen	MCA	75
Empoy	ME	65
Bibin	CSE	67

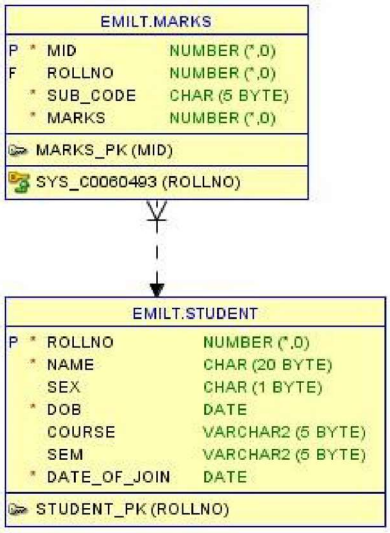
n. List all the students in the college other than CS Department
Query- select rollno,name from student where course !='CSE';

ROLLNO	NAME
2	Praveen
3	Empoy
4	Sourav

o. Count the number of students in each department whose mark in greater than 60
Query- select count(distinct s.rollno) as no_of_students,s.course from student s,marks m where s.rollno=m.rollno and marks>60 group by course;

NO_OF_STUDENTS	COURS
1	ME
1	CSE
2	MCA

DATA MODELER



RESULT

SQL queries are executed and output is obtained, thus CO1 and CO2 achieved.

Date:_____

EXPERIMENT:05
DATA MODELER FAMILIARIZATION

AIM

customer(cust_id,cust_name) primary key(cust_id). Item(item_id,item_name,price) primary key(item_id) Sale(bill_no,bill_date,cust_id,item_id,qty_sold) key(cust_id),foreign key(item_id).

a. Create the tables using data modeler . Insert around 10 records in each of the tables. primary key(bill_no),foreign

b. List all the bills for the current date with the customer name and item_no.

c. List the total bill detail with the quantity sold, price of the item and final amount.

d. List the details of the customer who have brought a product which has price>200.

e. Give a count of how many product have been brought by each customer.

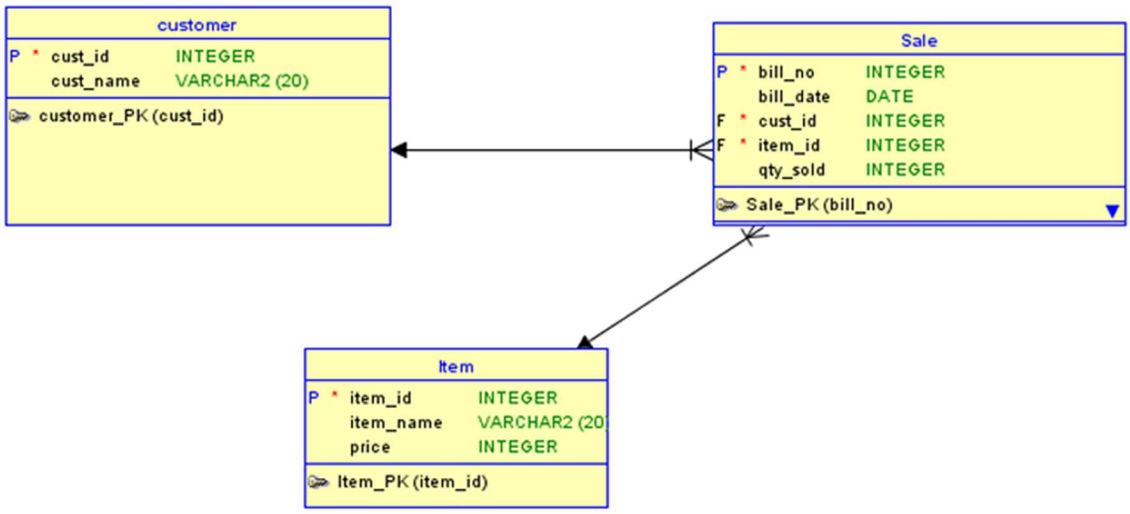
f. Give a list of product brought by a customer having cust_id 5.

g. List the item details which are sold as of today.

COURSE OUTCOME

CO1:Design database schema for a given real world problem-domain using standard design and modeling approaches.

TABLE CREATION



1.customer(cust_id,cust_name) primary key(cust_id).

Query- CREATE TABLE customer (cust_id VARCHAR2(5) NOT NULL,); cust_name VARCHAR2(25)

Query-ALTER TABLE customer ADD CONSTRAINT customer_pk PRIMARY KEY (cust_id);

2. Item(item_id,item_name,price) primary key(item_id)

Query-CREATE TABLE item (item_id VARCHAR2(5) NOT NULL, item_name VARCHAR2(25), price NUMBER);

Query-ALTER TABLE item ADD CONSTRAINT item_pk PRIMARY KEY (item_id);

3.Sale(bill_no,bill_date,cust_id,item_id,qty_sold) primary key(bill_no),foreign key(cust_id),foreign key(item_id).

Query-CREATE TABLE sale (bill_no VARCHAR2(5) NOT NULL, bill_date DATE, cust_id item_id VARCHAR2(5) NOT NULL, VARCHAR2(5) NOT NULL, qty_sold NUMBER);

Query-ALTER TABLE sale ADD CONSTRAINT sale_pk PRIMARY KEY (bill_no);

Query-ALTER TABLE sale ADD CONSTRAINT sale_customer_fk FOREIGN KEY (cust_id) REFERENCES customer (cust_id);

Query-ALTER TABLE sale ADD CONSTRAINT sale_item_fk FOREIGN KEY (item_id) REFERENCES item (item_id);

INSERTING VALUES

CUSTOMER TABLE

Query- insert into customer values(&cust_id,&cust_name);

Enter value for cust_id: ’1’

Enter value for cust_name: ‘amal’

Enter value for cust_id: ‘2’

Enter value for cust_name: ‘alin’

Enter value for cust_id: ‘3’

Enter value for cust_name: ‘alan’

ITEM TABLE

Query-insert into item values(&item_id,&item_name,&price);

Enter value for item_id: 'I101'

Enter value for item_name: 'eraser'

Enter value for price: 300

Enter value for item_id: 'I102'

Enter value for item_name: 'book'

Enter value for price: 200

Enter value for item_id: 'I103'

Enter value for item_name: 'pencil'

Enter value for price: 50

SALE TABLE

Query-insert into sale values(&bill_no,&bill_date,&cust_id,&item_id,&qty_sold);

Enter value for bill_no: '201'

Enter value for bill_date: '20-may-2024'

Enter value for cust_id: '1'

Enter value for item_id: 'I101'

Enter value for qty_sold: 2

Enter value for bill_no: '201'

Enter value for bill_date: '02-may-2024'

Enter value for cust_id: '2'

Enter value for item_id: 'I102'

Enter value for qty_sold: 3

QUERIES

b. List all the bills for the current date with the customer name and item_no.

Query-select s.bill_no,c.cust_name,i.item_id from sale s,customer c ,item i where c.cust_id=s.cust_id and i.item_id=s.item_id and s.bill_date=to_char(sysdate);

```
SQL> select cust_id,cust_name from sale natural join customer where bill_date=to_date('19-aug-2024');

   CUST_ID CUST_NAME
---------- --------------------
         3 aleena
         9 aida
        10 ann

SQL>
```

c.List the total bill detail with the quantity sold, price of the item and final amount.

Query-select s.bill_no,i.item_name,s.qty_sold,i.price,(i.price*s.qty_sold) as TOTAL_AMOUNT from sale s,item i where s.item_id=i.item_id;

```
SQL> select bill_no,cust_id,qty_sold,bill_date,item_id,price,(price*qty_sold) as total from sale natural join item;

   BILL_NO    CUST_ID   QTY_SOLD BILL_DATE    ITEM_ID      PRICE      TOTAL
---------- ---------- ---------- --------- ---------- ---------- ----------
       201          1          2 20-MAY-24        101         50        100
       202          2          3 02-MAY-24        102        100        300
       203          3          1 19-AUG-24        103       1000       1000
       204          4          4 18-AUG-24        104         10         40
       205          5          2 17-AUG-24        105         20         40
       206          6          3 16-AUG-24        106         15         45
       207          7          3 15-AUG-24        107        500       1500
       208          8          4 14-AUG-24        108        700       2800
       209          9          1 19-AUG-24        109       1700       1700
       210         10          2 19-AUG-24        110         60        120

10 rows selected.

SQL>
```

d.List the details of the customer who have brought a product which has price>200.

Query-select c.cust_name,i.item_name,i.price from customer c,item i,sale s where s.cust_id=c.cust_id and s.item_id=i.item_id and i.price>200;

```
SQL> select distinct cust_name,cust_id from customer natural join item where price >200;

CUST_NAME              CUST_ID
-------------------- ----------
albin                         4
adithya                       8
ashna                         7
alin                          1
amal                          5
alan                          2
ann                          10
amal                          6
aleena                        3
aida                          9

10 rows selected.
```

e. Give a count of how many product have been brought by each customer.

Query-select c.cust_id,c.cust_name,count(i.item_id),sum(qty_sold) from customer c,sale s,item i where s.cust_id=c.cust_id and s.item_id=i.item_id group by (c.cust_id,c.cust_name);

```
SQL> select count(*)as count from sale group by cust_id;

     COUNT
----------
         1
         1
         1
         1
         1
         1
         1
         1
         1
         1

10 rows selected.

SQL> slct count
```

f. Give a list of product brought by a customer having cust_id 5.

Query-select c.cust_id,c.cust_name,i.item_name,i.price from customer c,sale s,item i where s.cust_id=c.cust_id and s.item_id=i.item_id and c.cust_id='5';

```
SQL> select cust_id,item_name from sale natural join item where cust_id=5;

   CUST_ID ITEM_NAME
---------- --------------------
         5 pen

SQL>
```

g. List the item details which are sold as of today

. Query-select i.item_id,i.item_name,i.price from item i,sale s where s.item_id=i.item_id and bill_date=to_char(sysdate);

```
SQL> select * from sale natural join item where bill_date< to_date('19-aug-2024');

   ITEM_ID    BILL_NO BILL_DATE    CUST_ID   QTY_SOLD ITEM_NAME                PRICE
---------- ---------- --------- ---------- ---------- -------------------- ----------
       101        201 20-MAY-24          1          2 eraser                       50
       102        202 02-MAY-24          2          3 book                        100
       104        204 18-AUG-24          4          4 pencil                       10
       105        205 17-AUG-24          5          2 pen                          20
       106        206 16-AUG-24          6          3 ruler                        15
       107        207 15-AUG-24          7          3 bottle                      500
       108        208 14-AUG-24          8          4 shoe                        700

7 rows selected.
```

RESULT: SQL queries are executed and output is obtained,thus CO1 achieved.

Date:_____

EXPERIMENT-6

VIEW CREATION

AIM

Create a table student_data with the following fields: roll number(primary key), name, age, place. Create a view which contains only roll number and name.

a. List the data of the view.

b. Describe the view.

c. Insert a value into the view and list the view and table.

d. Update a value in the view and list the view and table.

e. Delete a value in the table and list the view and table.

f. Create another view which contains name and age column and check whether the above operations are possible in this view.

COURSE OUTCOME

CO2: Construct queries using SQL for database creation, interaction, modification, and updation.

TABLE CREATION

Create a table student_data with the following fields: roll number(primary key), name, age, place.

Query-create table student_data(rollno int primary key,name varchar(25),age int,place varchar(25));

INSERTING VALUES

STUDENT_DATA TABLE

Query- insert into student_data values(&rollno,&name,&age,&place); Enter value for rollno:

1 Enter value for name: 'aleena'

Enter value for age: 19

Enter value for place: 'Kottayam'

Enter value for rollno: 2

Enter value for name: 'alin'

Enter value for age: 20

Enter value for place: 'Trivandrum'

Enter value for rollno: 3

Enter value for name: 'Alan'

Enter value for age: 20

Enter value for place: 'Pala'

QUERIES

a. Create a view which contains only roll number and name.
Query-create view data as select rollno,name from student_data;

```
SQL> create table student_data(std_rollno int primary key,std_name varchar(20),std_age int,std_place varchar(20));

Table created.

SQL> create view student_view as select std_rollno,std_name from student_data;

View created.
```

b. List the data of the view.
Query-select * from data;

```
SQL> select*from student_data;

STD_ROLLNO STD_NAME                STD_AGE STD_PLACE
---------- -------------------- ---------- --------------------
         1 adithya
         2 alan
         3 alin
```

c. Describe the view.
Query-desc data;

```
SQL> describe student_view;
 Name                                                                                                  Null?    Type
 ----------------------------------------------------------------------------------------------------- -------- ------------------------------------
STD_ROLLNO                                                                                             NOT NULL NUMBER(38)
STD_NAME                                                                                                        VARCHAR2(20)
```

d. Insert a value into the view and list the view and table. Query-insert into data values(1,Aleena");
Query-Select * from data;

```
SQL> select*from student_view;

STD_ROLLNO STD_NAME
---------- --------------------
         1 aleena
         2 alan
         3 alin
```

e. Update a value in the view and list the view and table.
- update data set rollno=1 where name=’adithya';

```
SQL> update student_data set std_name ='adithya' where std_rollno=1;

1 row updated.

SQL> select*from student_data;

STD_ROLLNO STD_NAME                STD_AGE STD_PLACE
---------- -------------------- ---------- --------------------
         1 adithya
         2 alan
         3 alin
```

f. b. Delete a value in the table and list the view and table.
Query-delete from data where name='alin';

```
SQL> delete from student_data where std_rollno = 3;

1 row deleted.

SQL> select*from student_data;

STD_ROLLNO STD_NAME                STD_AGE STD_PLACE
---------- -------------------- ---------- --------------------
         1 adithya
         2 alan
```

```
SQL> select*from std_view;

   STD_AGE STD_NAME
---------- --------------------
           adithya
           alan
```

g. Create another view which contains name and age column and check whether the above operations are possible in this view.

Query-create view data_st as select name,age from student_data
Query-Desc data_st;

```
Commit complete.

SQL> describe std_view;
 Name                                                                                                  Null?    Type
 ----------------------------------------------------------------------------------------------------- -------- ------------------------------
 STD_AGE                                                                                                        NUMBER(38)
 STD_NAME                                                                                                       VARCHAR2(20)

SQL> select*from std_view;

   STD_AGE STD_NAME
---------- --------------------
         adithya
         alan
```

Insertion is not possible.

```
SQL> update student_data set std_name = 'aida'where std_rollno = 2;

1 row updated.

SQL> select*from std_view;

   STD_AGE STD_NAME
---------- --------------------
           adithya
           aida
```

```
SQL> select*from student_data;

STD_ROLLNO STD_NAME                STD_AGE STD_PLACE
---------- -------------------- ---------- --------------------
         1 adithya
         2 aida

SQL> commit;

Commit complete.
```

RESULT

SQL queries are executed and output is obtained, thus CO2 achieved.

Date:_____

EXPERIMENT-7
SEQUENCE CREATION

AIM

Create a table student_info with the following fields: roll number, name, mark. Create a sequence that will increment value by 1, start with 101, max value up to 200 and comes in a cycle manner.

a) Display the next value of sequence.
b) Display current value of sequence.
c) Alter the sequence by updating increment value by 2.
d) Insert values into table and use sequence to enter values in roll number field.

COURSE OUTCOME

CO2: Construct queries using SQL for database creation, interaction, modification, and updation.

TABLE CREATION

Create a table student_info with the following fields: roll number, name, mark.

Query- create table student_info(rollno int primary key,name varchar(25),mark int);

INSERTING VALUES

STUDENT_INFO TABLE

Query- insert into student_info values(&rollno,&name,&mark);

Enter value for rollno: 1
Enter value for name: 'Aleena'
Enter value for mark: 45

Enter value for rollno: 2
Enter value for name: 'Alan'
Enter value for mark: 40

Enter value for rollno: 3
Enter value for name: 'Alin'
Enter value for mark: 35

Query-Desc student_info;

```
SQL> select*from student_info;

    ROLLNO NAME                      MARK
---------- -------------------- ----------
         1 Aleena                       45
         2 Alan                         40
         3 Alin                         35
         4 Amal                         41
         5 Ashna                        47
```

SEQUENCE CREATION

Create a sequence that will increment value by 1, start with 101, max value up to 200 and comes in a cycle manner.

Query- create sequence dat_info increment by 1 start with 101 minvalue 101 maxvalue 200 cycle;

```
Sequence created.
```

QUERIES

a) Display the next value of sequence.
Query-select dat_info.nextval from dual;

```
SQL> select seq_info.nextval from dual;

   NEXTVAL
----------
       103

SQL> /

   NEXTVAL
----------
       104
```

b) Display current value of sequence.
Query-select dat_info.currval from dual;

```
SQL> select seq_info.currval from dual;

   CURRVAL
----------
       104
```

c) Alter the sequence by updating increment value by 2.
Query-alter sequence dat_info increment by 2;

```
       104

SQL> alter sequence seq_info increment by 2;

Sequence altered.

SQL> select seq_info.nextval from dual;

   NEXTVAL
----------
       106

SQL>
```

d) Insert values into table and use sequence to enter values in roll number field. **Query**- insert into student_info values(&rollno,&name,&mark);
Enter value for rollno: dat_info.nextval
Enter value for name: 'Elbi'
Enter value for mark: 89

```
SQL>  insert into student_info values(seq_info.nextval,'Adithya',85);

1 row created.

SQL> select*from student_info;

    ROLLNO NAME                          MARK
---------- -------------------- ----------
         1 Aleena                        45
         2 Alan                          40
         3 Alin                          35
         4 Amal                          41
         5 Ashna                         47
       101 Aleena                        89
       102 Adithya                       85

7 rows selected.
```

RESULT

SQL queries are executed and output is obtained, thus CO2 achieved.

Date:_____

EXPERIMENT -8

PL/SQL -PRODUCT OF TWO NUMBERS

AIM

Write a PL/SQL program to print the product of two numbers.

COURSE OUTCOME

C04 : Implement procedures, functions, and control structures using PL/SQL. (Cognitive Knowledge Level: Apply)

ALGORITHM

1. Start
2. Declare a, b, c as numbers
3. Read the value of a and b from the user
4. Multiply a and b and store the result in c
5. Print the value of c
6. End

PROGRAM

```
declare
        a number;
        b number;
        c number;
begin
        a:=&a;
         b:=&b;
        c:=a*b;
        dbms_output.put_line('product =' || c);
end;
```

OUTPUT

```
SQL> @C:\Users\student\Documents\plsql.1.txt
 11  /
Enter value for a: 5
old   6: a:=&a;
new   6: a:=5;
Enter value for b: 2
old   7: b:=&b;
new   7: b:=2;
product=10

PL/SQL procedure successfully completed.
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-9

PL/SQL -LARGEST AND SMALLEST NUMBERS

AIM

Write a PL/SQL program to find largest and smallest of three numbers.

COURSE OUTCOME

CO4 : Implement procedures, functions, and control structures using PL/SQL. (Cognitive Knowledge Level: Apply)

ALGORITHM

1. Start
2. Declare a, b, c as numbers
3. Read the value of a ,b and c from the user
4. If a is greater than b and a is greater than c, print 'Largest= ' followed by a
5. Else if b is greater than a and b is greater than c, print 'Largest= ' followed by b
6. Else print 'Largest= ' followed by c
7. If a is less than b and a is less than c, print 'Smallest= ' followed by a
8. Else if b is less than a and b is less than c, print 'Smallest= ' followed by b
9. Else print 'Smallest= ' followed by c
10. End

PROGRAM

```
declare
        a number;
        b number;
        c number;
        lar number;
        small number;
```

```
begin
        a:=&a;
        b:=&b;
        c:=&c;
        if   a>b   and    a>c   then
                lar:=a;
        elsif  b>a  and  b>c  then
                lar:=b;
        else
                lar:=c;
        end if;
        if   a<b   and    a<c   then
                small:=a;
        elsif  b<a  and  b<c  then
                small:=b;
        else
                small:=c;
        end if;
        dbms_output.put_line('smallest ='||small);
        dbms_output.put_line('largest='||lar);
end;
```

OUTPUT

```
Enter value for a: 2
old   8: a:=&a;
new   8: a:=2;
Enter value for b: 5
old   9: b:=&b;
new   9: b:=5;
Enter value for c: 1
old  10: c:=&c;
new  10: c:=1;
smallest =1
largest=5
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-10

PL/SQL -AREA OF CIRCLE AND SQUARE

AIM

To write a PL/SQL program to find area of circle and square

COURSE OUTCOME

CO4 : Implement procedures, functions, and control structures using PL/SQL. (Cognitive Knowledge Level: Apply)

ALGORITHM

1. Start
2. Declare r, s, ac, and asq as numbers
3. Read the value of radius from the user and store it in r
4. Read the value of side from the user and store it in s
5. Calculate the area of the circle using the formula ac = 3.14 * r * r
6. Calculate the area of the square using the formula asq = s * s
7. Print the value of ac as 'Area of circle= ' followed by ac
8. Print the value of asq as 'Area of square= ' followed by asq
9. End

PROGRAM

```
declare
        r number;
        s number;
        ac number;
        asq number;
begin
```

```
    r:=&radius;
    s:=&side;
    ac:=3.14*r*r;
    asq:=s*s;
    dbms_output.put_line('Area of circle= '||ac);
    dbms_output.put_line('Area of square= '||asq);
end;
```

OUTPUT

```
SQL> set serveroutput on
SQL> /
Enter value for radius: 3
old   8: r:=&radius;
new   8: r:=3;
Enter value for side: 4
old   9: s:=&side;
new   9: s:=4;
Area of circle = 28.26
Area of square = 16

PL/SQL procedure successfully completed.
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-11

PL/SQL -SUM OF FIRST N NATURAL NUMBERS

AIM

To write a PL/SQL program to find sum of first n natural numbers.

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

1. Start
2. Declare n and s as numbers
3. Read the value of n from the user
4. Initialize s to 0
5. Loop from i = 1 to n
6. Add i to s
7. End loop
8. Print the value of s as 'Sum= ' followed by s
9. End

PROGRAM

```
declare
a number;
N number;
s number;
begin
s:=0;
n:=&n;
```

```
for i in 1..n loop
 s:=s+i;

end loop;


dbms_output.put_line('Sum is :');
dbms_output.put_line(s);

end;
```

OUTPUT

```
Enter value for n: 5
old   8: n:=&n;
new   8: n:=5;
Sum is :
15

PL/SQL procedure successfully completed.
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved

Date:_____

EXPERIMENT-12

PL/SQL - SEQUENCE CREATION

AIM

To write a PL/SQL program to obtain the sequence 1, 4, 9, 16...

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

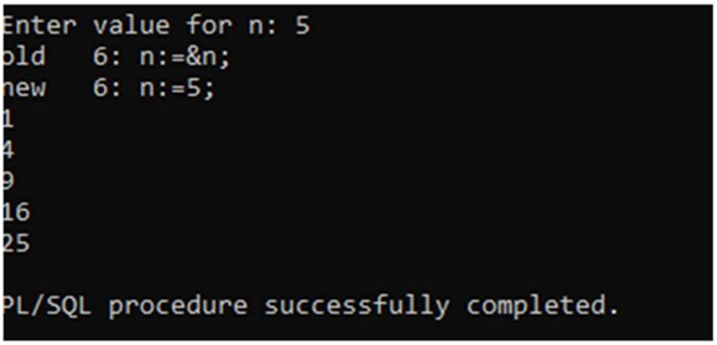
1. Start
2. Declare n and s as numbers
3. Read the value of n from the user
4. Initialize s to 0
5. Loop from i = 1 to n
6. Calculate the square of i and store it in s
7. Print the value of p
8. End loop
9. End

PROGRAM

```
declare
   n number;
   i number;
   s number:=0;
begin
   n:=&n;
```

```
    for i in 1..n loop
    s:=i*i;
dbms_output.put_line(s);
 end loop;
end;
```

OUTPUT



RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved

Date:_____

EXPERIMENT-13

PL/SQL -REVERSE A GIVEN STRING

AIM

To write a PL/SQL program to reverse a given string.

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

1. Start
2. Declare str, len, and rev as variables of type varchar and number respectively
3. Read the value of str from the user
4. Calculate the length of str and store it in len
5. Initialize rev to an empty string
6. Loop from i = len to 1 with a step of -1
7. Extract the i-th character of str using the substr function and append it to rev
8. End loop
9. Print the value of reversed string, rev
10. End

PROGRAM

```
Declare
    str varchar(8);
    i number;
   len number;
   rev varchar(8):='';
```

```
begin
  str:='&str';
  len:=length(str);

 for i in reverse 1..len loop
rev:=rev||substr(str,i,1);
end loop;
dbms_output.put_line('Reversal:'|| rev);
 end;
```

OUTPUT

```
Enter value for str: hello
old   7:  str:='&str';
new   7:  str:='hello';
Reversal:olleh

PL/SQL procedure successfully completed.
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved

Date:_____

EXPERIMENT-14

PL/SQL -ARMSTRONG OR NOT

AIM

Write a PL/SQL program to check given number is Armstrong or not.

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

1. Start
2. Declare s, r, p, q, and len as numbers and s is initialized to 0
3. Read the value of p from the user and store it in p
4. Store the value of p in q
5. Calculate the length of p and store it in len
6. Loop while p is greater than 0
7. Calculate the remainder of p divided by 10 and store it in r
8. Add r raised to len to s
9. Divide p by 10 and truncate the result to an integer
10. End loop
11. If q is equal to s, print 'Armstrong'
12. Else, print 'Not Armstrong'
13. End

PROGRAM

```
declare
        s number:=0;
        r number;
        p number;
        q number;
        len number;
begin

        p:=&p;
        q:=p;
```

```
        len:=length(to_char(p));
        while p>0
        loop
                r:=mod(p,10);
                s:=s+power(r,len);
                p:=trunc(p/10);
        end loop;
        if q=s then
                dbms_output.put_line('Armstrong');
        else
                dbms_output.put_line('Not Armstrong');
        end if;
end;
```

OUTPUT

```
SQL> /
Enter value for number: 371
old   8: n:=&number;
new   8: n:=371;
371 is armstrong

PL/SQL procedure successfully completed.

SQL> /
Enter value for number: 234
old   8: n:=&number;
new   8: n:=234;
234 is not armstrong

PL/SQL procedure successfully completed.
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-15

PL/SQL -SALARY INCREMENT

AIM

An employee is given 25% increase in salary if salary is above Rs. 25000 and 20% increase in salary if salary is above Rs. 30000. Write a PL/SQL program to calculate new salary.

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

1. Start
2. Declare two variables newsal and salary of type number and initialise newsal to 0.
3. Assign the value to the variable salary.
4. If the value of salary is greater than 25000, then calculate the new salary by adding 25% of the current salary to it. Assign the result to the variable newsal.
5. Else if the value of salary is greater than 30000, then calculate the new salary by adding 20% of the current salary to it. Assign the result to the variable newsal.
6. Else, assign the value of salary to the variable newsal.
7. Print the value of new salary
8. End

PROGRAM

```
declare
        newsal number:=0;
        salary number;
begin
        salary:=&salary;
        if salary>25000 then
                newsal:=salary+(0.25*salary);
        elsif salary>30000 then50
```

```
            newsal:=salary+(0.20*salary);
        else
            newsal:=salary;
        end if;
        dbms_output.put_line('New salary is: '||newsal);
end;
```

OUTPUT

```
SQL> set serveroutput on
SQL> /
Enter value for sal: 25001
old   4: sal:=&sal;
new   4: sal:=25001;
Updated salary:31251.25

PL/SQL procedure successfully completed.

SQL> /
Enter value for sal: 30001
old   4: sal:=&sal;
new   4: sal:=30001;
Updated salary:36001.2

PL/SQL procedure successfully completed.
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-16
PL/SQL -ODD AND EVEN NUMBERS

AIM

To write a PL/SQL program to insert first 15 odd numbers into a table ODD and first 15 even numbers into a table EVEN.

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

1. Start
2. Use a for loop to iterate over the numbers from 1 to 30.
3. Check if the current number is even or odd using the mod() function.
4. If the number is even, insert it into the EVEN table.
5. If the number is odd, insert it into the ODD table.
6. Print a message to the console indicating that the numbers have been inserted into the ODD and EVEN tables successfully.
7. Stop

PROGRAM

```
begin
        for i in 1..30
        loop
        if mod(i,2)=0 then
                insert into EVEN values(i);
        else
                insert into ODD values(i);
        end if;
        end loop;
        dbms_output.put_line('Numbers have been inserted into the ODD and EVEN tables
        successfully');
end;
```

OUTPUT

Even Numbers

```
SQL> select * from EVEN;

      EVEN
----------
         0
         2
         4
         6
         8
        10
        12
        14
        16
        18
        20

      EVEN
----------
        22
        24
        26
        28
        30
```

Odd Numbers

```
SQL> select * from ODD;

       ODD
----------
         1
         3
         5
         7
         9
        11
        13
        15
        17
        19
        21

       ODD
----------
        23
        25
        27
        29
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT 17

IMPLICIT CURSOR – SALARY UPDATION

AIM

Write a PL/SQL program to update salary of Sindhu by 30% if she is earning salary>10000, otherwise update by 20% if salary>8000, otherwise update by 10%. (Table name: income(ename, salary).

COURSE OUTCOME

C03 Design and implement triggers and cursors. (Cognitive Knowledge Level: Apply)

ALGORITHM

1. Start

2. Declare a variable i of type income%rowtype.

3. Select the salary of the employee named Sindhu from the income table and assign it to the variable i.salary.

4. If the value of i.salary is greater than 10000, then calculate the new salary by adding 30% of the current salary to it. Assign the result to the variable i.salary.

5. Else if the value of i.salary is greater than 8000, then calculate the new salary by adding 20% of the current salary to it. Assign the result to the variable i.salary.

6. Else, calculate the new salary by adding 10% of the current salary to it. Assign the result to the variable i.salary.

7. Update the income table with the new salary of the employee named Sindhu.

8. Stop

//TABLE CREATION

Query: create table income(name varchar(20), salary int);

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
NAME                                               VARCHAR2(30)
SALARY                                             NUMBER(38)
```

//INSERTING VALUES INTO INCOME

Query: insert into income values(‘&name’, &salary);

Query: select * from income;

```
SQL> insert into income values('Sindhu', 11000);

1 row created.

SQL> insert into income values('Panicker', 9450);

1 row created.

SQL> insert into income values('Alin', 7200);

1 row created.
```

PROGRAM

```
declare
        i income%rowtype;
begin
select salary into i.salary from income where name=’Sindhu’;
        if(i.salary>10000) then
        i.salary:=i.salary+(0.30*i.salary);
        elsif(i.salary>8000) then
        i.salary:=i.salary+(0.20*i.salary);
        else
        i.salary:=i.salary+(0.10*i.salary)
                end if;
        update income set salary=i.salary where name=’Sindhu’;
end;
```

OUTPUT

```
SQL> select * from income;

NAME                      SALARY
-------------------- ----------
Sindhu                     14300
Panicker                    9450
Alin                        7200
```

RESULT

PL/SQL program executed and the output was obtained, thus CO3 achieved.

Date:_____

EXPERIMENT 18

IMPLICIT CURSOR – SALARY CHECKING AND ROLLBACK

AIM

To write a PL/SQL program to update salary of all employees by 20%. If total salary>100000 then rollback else commit.

COURSE OUTCOME

CO3 Design and implement triggers and cursors. (Cognitive Knowledge Level: Apply)

ALGORITHM

1. Start.
2. Declare a variable tsalary of type income.salary%type.
3. Update the salary column of the income table by adding 20% of the current salary to it.
4. Select the sum of all salaries from the income table and assign it to the variable tsalary.
5. If the value of tsalary is greater than 100000, then rollback the transaction.
6. Else, commit the transaction.
7. Stop.

//TABLE CREATION

Query: create table income(name varchar(30),salary int);

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------
NAME                                               VARCHAR2(30)
SALARY                                             NUMBER(38)
```

//INSERTING VALUES INTO INCOME

Query: insert into income values('&name', &salary);

Query: select * from income;

```
SQL> select * from income;

NAME                      SALARY
-------------------- ----------
Sindhu                     14300
Panicker                    9450
Alin                        7200
```

PROGRAM

```
declare
        tsalary income.salary%type;
begin
update income set salary=salary+(0.20*salary);
select sum(salary) into tsalary from income;
if tsalary>100000 then
rollback;
else
commit;
end if;
end;
```

OUTPUT

```
SQL> select * from income;

NAME                      SALARY
-------------------- ----------
Sindhu                     17160
Panicker                   11340
Alin                        8640
```

RESULT

PL/SQL program executed and the output was obtained, thus CO3 achieved.

Date:_____

EXPERIMENT 19

IMPLICIT CURSOR – EXCEPTION HANDLING

AIM

Create a table student(rollno, stud_name,sessionals, univ_mark). If the sessionals+univ_mark>150, raise an error message. Also handle all the possible exceptions.

COURSE OUTCOME

CO3 Design and implement triggers and cursors. (Cognitive Knowledge Level: Apply)

ALGORITHM

1.Start

2. Declare c as an exception, s as an integer, t as an integer, and sval as a record of type student%rowtype

3. Read the value of rollno from the user and store it in s

4. Select sessionals and univ_mark into sval.sessionals and sval.univ_mark respectively from student where rollno = s

5. Calculate the sum of sessionals and univ_mark and store it in t

6. If t is greater than 150, raise the exception c

7. Else, print 'NO ERROR'

8. Handle the exception c by printing 'ERROR! SESSIONALS+UNIV MARK CANNOT BE GREATER THAN 150'

9. Handle the exception NO_DATA_FOUND by printing 'ERROR! NO DATA FOUND'

10. Handle the exception OTHERS by printing 'ERROR! OPERATION NOT POSSIBLE'

11. Stop

//TABLE CREATION

Query: create table student(rollno int, std_name varchar(20), sessionals int, univ_mark int);

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------
ROLLNO                                             NUMBER(38)
STD_NAME                                           VARCHAR2(20)
SESSIONALS                                         NUMBER(38)
UNIV_MARK                                          NUMBER(38)
```

//INSERTING VALUES INTO STUDENT

Query: insert into student values(&rollno,'&std_name',&sessionals,&univ_mark);

Query: select * from student;

```
SQL> select * from students;

    ROLLNO STD_NAME             SESSIONALS UNIV_MARKS
---------- -------------------- ---------- ----------
         1 Alin                         15         25
         2 Sidhu                        10         22
         3 Allan                        45        134
```

PROGRAM

```
declare
        c exception;
        s int;
        t int;
        sval student%rowtype;
begin
        s:=&rollno;
        select sessionals, univ_mark into sval.sessionals, sval.univ_mark from student
        where rollno=s;
        t:=sval.sessionals+sval.univ_mark;
        if t>150 then
                raise c;
        else
                dbms_output.put_line('NO ERROR');
```

```
        end if;
exception
        when c then
                dbms_output.put_line('ERROR! SESSIONALS+UNIV MARK CANNOT BE
                GREATER  THAN 150');
        when NO_DATA_FOUND then
                dbms_output.put_line('ERROR! NO DATA FOUND');
        when OTHERS then
                dbms_output.put_line('ERROR! OPERATION NOT POSSIBLE');
end;
```

OUTPUT

```
Output:

Enter value for rollno: old  13: s:=&rollno;
new  13: s:=3;
ERROR! SESSIONALS+UNIV MARK CANNOT BE   GREATER  THAN 150
```

RESULT

PL/SQL program executed and the output was obtained, thus CO3 achieved.

Date:_____

EXPERIMENT-20

EXPLICIT CURSOR-MESS FEE INCREMENT

AIM:

Create a hostel mess database with fields(stud_no, name, messfee, veg/nonveg). Write a PL/SQL program to increase the mess fee of vegetarians by 10% and non vegetarians by 20%. Also create tables vegetarian and non_vegetarian which includes fields: stud_no, name, raise_in_fee and date on which raise was given. Insert values into these tables through PL/SQL program.

COURSE OUTCOME:

CO3: Analyze stored programming concepts using cursors and triggers.

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM:

1. Start
2. Declare a cursor c that selects all the columns from the table mess
3. Declare a variable i of the same type as a row in the table mess
4. Open the cursor
5. Loop over the records in the cursor
6. Fetch the next record from the cursor c and store it in i
7. If there are no more records, exit the loop
8. Check the value of the column veg_nonveg in i
9. Insert the student number, name, mess fee with 20% extra charge, and the current date into the table nonvegetarians
10. Update the mess fee in i by adding 20% to it
11. Update the mess fee in the table mess for the student number in i
12. Otherwise, Insert the student number, name, mess fee with 10% extra charge, and the current date into the table vegetarians
13. Update the mess fee in i by adding 10% to it
14. Update the mess fee in the table mess for the student number in i
15. Close the cursor

PROGRAM:

```
declare

        i hostel_mess%rowtype;

        fees number;

        cursor c is select * from hostel_mess;
```

```
begin
        open c;
        loop
                fetch c into i;
                exit when c%notfound;
                if i.veg_nonveg = 'veg' then
                        fees := i.messfee+(i.messfee*0.1);
                        insert into vegetarian values(i.stud_no,i.name,i.messfee*0.1,sysdate);
                elsif i.veg_nonveg = 'nonveg' then
                        fees := i.messfee+(i.messfee*0.2);
                        insert into vegetarian values(i.stud_no,i.name,i.messfee*0.2,sysdate); end
                if;
                update hostel_mess set messfee=fees where stud_no=i.stud_no;
end loop;
close c;
```

OUTPUT:

create table hostel_mess (stud_no number primary key, name varchar(50), messfee number, veg_nonveg varchar(10));

desc hostel_mess;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
STUD_NO                                   NOT NULL NUMBER
NAME                                               VARCHAR2(50)
MESSFEE                                            NUMBER
VEG_NONVEG                                         VARCHAR2(10)
```

create table vegetarian (stud_no number primary key, name varchar(50), raise_in_fee number, raise_date date);

desc vegetarian;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
STUD_NO                                   NOT NULL NUMBER
NAME                                               VARCHAR2(50)
RAISE_IN_FEE                                       NUMBER
RAISE_DATE                                         DATE
```

create table non_vegetarian (stud_no number primary key, name varchar2(50), raise_in_fee number, raise_date date);

desc non_vegetarian;

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STUD_NO                                   NOT NULL NUMBER
 NAME                                               VARCHAR2(50)
 RAISE_IN_FEE                                       NUMBER
 RAISE_DATE                                         DATE
```

insert into hostel_mess values (1, 'Antony', 3000, 'veg');

insert into hostel_mess values (2, 'Aravind', 3200, 'nonveg');

insert into hostel_mess values (3, 'Anand', 2800, 'veg');

insert into hostel_mess values (4, 'Angela', 3300, 'nonveg');

select * from hostel_mess;

```
   STUD_NO NAME                                               MESSFEE VEG_NONVEG
---------- -------------------------------------------------- ---------- ----------
         1 Antony                                               3000 veg
         2 Aravind                                              3200 nonveg
         3 Anand                                                2800 veg
         4 Angela                                               3300 nonveg
         5 Ria                                                  3100 veg
         6 Josh                                                 2900 nonveg
```

select * from hostel_mess;

```
   STUD_NO NAME                                               MESSFEE VEG_NONVEG
---------- -------------------------------------------------- ---------- ----------
         1 Antony                                               3300 veg
         2 Aravind                                              3840 nonveg
         3 Anand                                                3080 veg
         4 Angela                                               3960 nonveg
         5 Ria                                                  3410 veg
         6 Josh                                                 3480 nonveg

6 rows selected.
```

select * from vegetarian;

```
   STUD_NO NAME                                           RAISE_IN_FEE RAISE_DAT
---------- -------------------------------------------------- ------------ ---------
         1 Antony                                                  300 07-OCT-24
         3 Anand                                                   280 07-OCT-24
         5 Ria                                                     310 07-OCT-24
```

select * from non_vegetarian;

```
   STUD_NO NAME                                           RAISE_IN_FEE RAISE_DAT
---------- -------------------------------------------------- ------------ ---------
         2 Aravind                                                 640 07-OCT-24
         4 Angela                                                  660 07-OCT-24
         6 Josh                                                    580 07-OCT-24
```

RESULT:

PL/SQL program executed and the output was obtained,thus CO3 and CO4 achieved

Date:_____

EXPERIMENT-21

EXPLICIT CURSOR-MODERATION

AIM

Create a table T1 having 3 fields(rollno, univ_mark and sessionals). Write a PL/SQL program to do the following: If sessionals is in between 30 and 34, then give necessary moderation so that it comes upto 35. If univ_mark+sessionals>75, then insert those tuples into another table T2

COURSE OUTCOME

CO3: Analyze stored programming concepts using cursors and triggers.

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

1. Start
2. Declare a cursor c that selects all the columns from the table T1
3. Declare a variable t of the same type as a row in the table T1
4. Open the cursor
5. Check if the cursor is open
6. Loop over the records in the cursor
7. Fetch the next record from the cursor and store it in t
8. If there are no more records, exit the loop
9. Check the value of the column sessionals in t
10. If t.sessionals is between 30 and 34, Update the sessionals to 35 in the table T1 for the rollno in t
11. Check the value of the column univ_mark plus the column sessionals >75, Insert the rollno, univ_mark, and sessionals into the table T2
12. Close the cursor

PROGRAM:

```
declare
        t ti%rowtype;
        cursor c is select * from tl;
begin
        open c;
        loop
                fetch c into t;
                exit when c%notfound;
                if (t.sessionals between 30 and 30) then
                        t. sessionats:=35;
                        update tl set sessionals =t.sessionals where rollno=t.rollno;
                end if;
                if (t.univ_mark + t.sessionals > 75) then
                        insert into t2 values (t.rottno, t.univ_mark, t.sessionals);
                end if;
        end loop;
        close c;
end;
```

OUTPUT:

create table T1(rollno number(3) primary key,univ_mark number(3),sessionals number(3));

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------
ROLLNO                                    NOT NULL NUMBER(3)
UNIV_MARK                                          NUMBER(3)
SESSIONALS                                         NUMBER(3)
```

insert into T1 values(1,50,45); insert into T1 values(2,30,35); insert into T1 values(3,20,36); select * from T1;

```
SQL> select * from T1;

    ROLLNO  UNIV_MARK SESSIONALS
---------- ---------- ----------
         1         50         45
         2         30         35
         3         20         36
```

create table T2(rollno number(3) primary key,univ_mark number(3),sessionals number(3));

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------
ROLLNO                                    NOT NULL NUMBER(3)
UNIV_MARK                                          NUMBER(3)
SESSIONALS                                         NUMBER(3)
```

```
SQL> select * from T1;

    ROLLNO  UNIV_MARK SESSIONALS
---------- ---------- ----------
         1         50         45
         2         30         35
         3         20         36

SQL> select * from T2;

    ROLLNO  UNIV_MARK SESSIONALS
---------- ---------- ----------
         1         50         45
```

RESULT

PL/SQL program executed and the output was obtained,thus CO3 and CO4 achieved

Date:_____

EXPERIMENT-22
FUNCTION -STUDENT MARK

AIM

Write a function which accepts the reg_no and print the total marks. The student table has thefields: reg_no, name, physics_mark, chemistry_mark and maths_mark.

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

1. Start
2. Define a function f that takes a number r as input and returns a number function f(r)
3. Declare a variable t as number to store the result
4. Calculate sum of all subjects from table student1 and assign it to the variable t
5. Return the value of t
6. End function

Function Call

1. Declare variables r and t as number to store the input and output
2. Prompt the user to enter a value for r
3. Call the function f with r as the argument and assign the result to t
4. Print the value of t to the output as sum

PROGRAM

Function Definition

```
create or replace function f(no number) return number is
        p student4.phy%type;
        c studentLl.chem%type;
        m student4.math%type;
        tot number(5) ;
BEGIN
        select phy , chem,math into p, c, m from student4 where regno=no;
        tot : =p+c+m ;
        return (tot) ;
END
```

Function Call

```
DECLARE
        r student4.regno%type;
        p number(10);
BEGIN
        r:=&r;
        p:=f(r);
        dbms_output.put_line('total mark'||p);
END;
```

OUTPUT:

create table student1(reg_no int,name varchar(20),physics_mark number,chemistry_mark number,maths_mark number);
desc student1;

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
REG_NO                                             NUMBER(38)
NAME                                               VARCHAR2(20)
PHYSICS_MARK                                       NUMBER
CHEMISTRY_MARK                                     NUMBER
MATHS_MARK                                         NUMBER
```

insert into student1 values(101,'Abraham',74,86,92);
insert into student1 values(102,'Alana',89,79,94);
insert into student1 values(103,'Benjamin',92,84,95);
select * from student1;

```
   REG_NO NAME                PHYSICS_MARK CHEMISTRY_MARK MATHS_MARK
--------- ------------------- ------------ -------------- ----------
      101 Abraham                       74             86         92
      102 Alana                         89             79         94
      103 Benjamin                      92             84         95
      104 Daniel                        80             78         85
      105 Faizzi                        86             81         91
```

```
Function created.
```

```
Enter value for r: 103
old   5:        r:=&r;
new   5:        r:=103;
sum=271
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-23

FUNCTION-SUM OF FIRST N EVEN NUMBERS

AIM

Write a function to find sum of first N even no:s

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM

1. Start
2. Define a function f that takes a number r as input and returns a number function f(r)
3. Declare and initialize a variable t as number to store the result
4. Set t to 0
5. Loop from 0 to (r*2)-1
    5.1 Check if i is even, add i to t
6. End loop
7. Return the value of t
8. End function

Function Call

1. Declare variables r and t as number to store the input and output
2. Prompt the user to enter a value for r
3. Call the function f with r as the argument and assign the result to t
4. Print the value of t to the output as sum

PROGRAM:

Function Definition

```
create or replace function f1(no in number) return number is
        s number:=0;
        i number:=2;
        ab number:=l;
begin
        while(ab<=no)
                loop
                        s:=s+i;
                        i:=i+2;
                        ab:=ab+1;
                end loop;
        return(s);
end;
```

Function call

```
DECLARE
        no number(10):=&no
        s number(l0);
BEGIN
        s:=f1(no);
        dbms_output.put_line(' sum'||s);
END;
```

OUTPUT

```
SQL> set serveroutput on
SQL> /
Enter value for limit: 6
old   5: n:=&limit;
new   5: n:=6;
Sum=12

PL/SQL procedure successfully completed.
```

RESULT :

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-24

PROCEDURE-GRADE DISPLAY

AIM

Write a PL/SQL program to display the grade of a particular student from student database. Use a stored procedure to display the grade.

TOTAL MARK GRADE

80-100 A

70-80 B

50-70 C

<50 Fail

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions

ALGORITHM:

1. Start
2. Define a procedure p that takes a number r as input procedure p(r)
3. Declare variables g as string and t as number to store the grade and the total mark 4. Calculate the sum of marks of all subjects from table student1 of each student and store it in variable t
5. Divide t by 3 to get the average mark
6. Assign a grade based on the average mark
7. If t is between 80 and 100, g = "A"
8. Else if t is between 70 and 80, g = "B"
9. Else if t is between 50 and 70, g = "C" 10.Otherwise, g = "F"
11. Print the grade to the output
12. End procedure

PROGRAM:

```
create or replace procedure pr(no number) is
        tot number;
begin
        select mark into tot from student5 where rno=no;
        if(tot>100)then
                dbms_output.put_line('A');
        elsif(tot>70 and tot<100) then
                dbms_output.put_line('B');
        elsif(tot>50 and tot<70) then
                dbms_output.put_line('C');
        else
                dbms_output.put_line('Fail');
        end if;
end;
```

OUTPUT:

```
SQL> select * from student5;

       RNO NAME                 MARK
---------- ---------- ----------
         1 harry                81
         2 fred                 60
```

```
Procedure created.
```

```
SQL> DECLARE
  2   r number(10);
  3  BEGIN
  4   r:=&r;
  5   pr(r);
  6  END;
  7  /
Enter value for r: 1
old   4:  r:=&r;
new   4:  r:=1;
B

PL/SQL procedure successfully completed.

SQL> /
Enter value for r: 2
old   4:  r:=&r;
new   4:  r:=2;
C

PL/SQL procedure successfully completed.
```

RESULT:

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-25

PACKAGE

AIM

Create a Package to compute the total mark through function and derive the grade using procedure.

COURSE OUTCOME

CO4: Apply PL-SQL concepts such as various control structures, creation of procedures and functions.

ALGORITHM:

1. Start
2. Define a package stud that contains a procedure p and a function f package stud
3. Declare a procedure p that takes a number r as input
4. Declare variables g as string and t as number to store the grade and the total mark
5. Calculate the sum of marks of all subjects of each student from table student1 and store it into variable t
6. Divide t by 3 to get the average mark
7. Assign a grade based on the average mark
8. If t is between 80 and 100, g = "A"
9. Else if t is between 70 and 80, g = "B"
10. Else if t is between 50 and 70, g = "C"
11. Otherwise, g = "F"
12. Print the grade to the output
13. End package
14. Declare a function f that takes a number r as input and returns a number
15. Declare a variable t as number to store the result
16. Calculate the sum of marks of all subjects of each student from table student1 and store it into variable t

17. Return t
18. End function
19. End procedure

PROGRAM:

Package creation

```
create or replace package s as
        procedure p(r in number);
        function f(r in number)
        return number;
end;
```

Package Body

```
create or replace package body s as
        procedure p(r in number) is
        g varchar(1);
        t number;
begin
        select physics+chemistry+maths into t from student5 where r=rollno;
        t:=t/3;
        if t between 80 and 100 then
                g:='a';
        elsie t between 70 and 80 then
                g:='b';
        elsie t between 50 and 70 then
                g:='c';
        else
                g:='f';
        end if;
                dbms_output.put_line('grade is: '||g);
        end p;
```

```
    function f(r in number) return number is t number;
    begin
        select physics+chemistry+maths into t from student5 where r=rollno;
        return t;
    end f;
end;
```

OUTPUT:

```
Package created.
```

```
Package body created.
```

Package Calling

```
declare
    r number;
    t number;
    begin
    r:=&rollno;
    t:=s.f(r);
    s.p(r);
    dbms_output.put_line('total mark: '||t);
end;
```

```
Enter value for rollno: 1
old   5: R:=&ROLLNO;
new   5: R:=1;

PL/SQL procedure successfully completed.
```

RESULT

PL/SQL program executed and the output was obtained, thus CO4 achieved.

Date:_____

EXPERIMENT-26

TRIGGER- ACCOUNT TABLE

AIM

To create an account table(acc_no, cname, balance, branch_ name), loan table(loan_no, amt, branch_name), borrower table(cname, loan_no). Create a trigger to perform the following operations: Whenever the balance becomes negative, create a loan in the amount of overdraft. The loan_no is given same as acc_no.

COURSE OUTCOME

CO3: Analyze stored programming concepts using cursors and triggers.

ALGORITHM

1. Create a table account with fields accno, cname, bal and branchname.
2. Create a table loan with fields loanno, amount, branchname.
3. Create a table borrower with fields cname, loanno.
4. Create a trigger t which executes after update or insert on each row of account table.
5. Insert into loan table new.accno,new.bal-1 and new.branchname.
6. Insert into borrower table new.cname, new.loanno
7. End the trigger.
8. Insert values into the account table
9. End.

SQL> create table account(accno number(10), cname char(10), balance number(10), branch char(10));

Table created.

SQL> create table loan(loanno number(10), amt number(10), branch char(10));

Table created.

SQL> create table borrower(cname char(10), loanno number(10));

Table created.

SQL> insert into account values(1, 'tibin', 200, 'ktpna');

1 row created.

PROGRAM

```
create or replace trigger loaning
        after update on account for each row
        when (new.balance<0)
begin
        insert into loan values(:oId.acc_no,:new.balance*-1,:old.branch_name);
        insert into borrower values(:old.cname,: old.acc_no);
end ;
```

OUTPUT

```
SQL> update account set balance=-500 where acc_no=3;

1 row updated.
```

```
SQL> select * from borrower;

CNAME                          LOAN_NO
------------------------------ ----------
JJJ                                 3

SQL> select * from loan;

   LOAN_NO        AMT BRANCH_NAME
---------- ---------- --------------------------------------------------
         3        500 LLL
```

RESULT

PL/SQL program executed and the output was obtained, thus CO3 achieved

Date:_____

EXPERIMENT-27

TRIGGER- AUDIT SYSTEM

AIM

Create a transparent audit system for a table clientmaster. The system has to keep track of records that have been removed or modified and when they have been removed or modified. Table details are given below: AuditClient: name, bal_due, operation, Op_date

COURSE OUTCOME

CO3: Analyze stored programming concepts using cursors and triggers.

ALGORITHM

1. Create a table cli with fields cno, name, adrs and due.
2. Create a table auditcli with fields name, baldue, op, date.
3. Create a trigger trig which executes after update or delete on each row of cli table.
4. If updating, insert into auditcli table :old.name, :old.due, 'Update',sysdate
5. If deleting, insert into auditcli :old.name,:old.due,'Delete',sysdate;
6. End the trigger.
7. Insert values into the account table
8. End.

PROGRAM

```
create or replace trigger t2
        after update or delete on clientmaster
        for each row
begin
```

```
    if updating then
        Insert into audit_client values(:old.name,:new.bal_due,'UPDATE',SYSDATE);
    else
        Insert into audit_client values(:old.name,:old.bal_due,'DELETE',SYSDATE);
    end if;
end;
```

OUTPUT

```
SQL> SELECT * FROM CLIENTMASTER;

      C_NO NAME                           ADDRESS                          BAL_DUE
---------- ------------------------------ ------------------------------ ----------
         1 JACKIE                         BOLL                                150
         2 CHAN                           BOLLLA                              250
         3 FACK                           FEAR STREET                         250
         4 HIAN                           LA                                 1250
```

```
SQL> UPDATE CLIENTMASTER SET BAL_DUE=5000 WHERE C_NO=1;

1 row updated.
```

DELETE FROM CLIENTMASTER WHERE C_NO=4;

```
1 row deleted.
```

DELETE FROM CLIENTMASTER WHERE C_NO=2;

```
1 row deleted.
```

```
SQL> SELECT * FROM AUDITCLIENT;

NAME                           BAL_DUE OPERATION                      OP_DATE
------------------------------ ---------- ------------------------------ ---------
JACKIE                           5000 UPDATE                         10-OCT-24
HIAN                             1250 DELETE                         10-OCT-24
```

RESULT

PL/SQL program executed and the output was obtained, thus CO3 achieved

Date:_____

EXPERIMENT-28

TRIGGER-RAISING ERRORS

AIM

To create a table with two number fields a and b and to write a trigger that satisfies the following condition: a. a+b>75. b. c. If value of b is changed, it should not be changed to a smaller value. Also the tuples that violate these conditions should not be entered

COURSE OUTCOME

CO3: Analyze stored programming concepts using cursors and triggers.

ALGORITHM

1. Create a table with fields a,b.
2. Create a trigger test_trigger which executes before insert or update on each row of the test table.
3. If a+b<=75, then raise exception1
4. If new.b < old.b then raise exception2
5. End the trigger
6. End

PROGRAM

```
create or replace trigger t3 before update or insert on trig3 for each row
begin
        if :new.a + :new.b<75 then
                raise_application_error(-20001,'Invalid sum');
        end if;
        if :new.b<:old.b then
```

```
        raise_application_error(-20002,'B is lesser');
    end if;
end;
```

OUTPUT

```
SQL> insert into trig3 values(30,15);
insert into trig3 values(30,15)
            *
ERROR at line 1:
ORA-20001: Invalid sum
ORA-06512: at "JIMMY.T3", line 3
ORA-04088: error during execution of trigger 'JIMMY.T3'
```

```
SQL> insert into trig3 values(30,45);

1 row created.

SQL> select * from trig3;

         A          B
---------- ----------
        30         45
```

```
SQL> update trig3 set b=40 where b=45;
update trig3 set b=40 where b=45
       *
ERROR at line 1:
ORA-20001: Invalid sum
ORA-06512: at "JIMMY.T3", line 3
ORA-04088: error during execution of trigger 'JIMMY.T3'
```

RESULT

PL/SQL program executed and the output was obtained, thus CO3 achieved.

Date:_____

EXPERIMENT-29

CRUD OPERATIONS IN MONGO DB

AIM

To perform the following CRUD operations:

1) CREATE

2) INSERT

3) READ

4) UPDATE

5) DELETE

COURSE OUTCOME

CO5:Perform CRUD operations in NoSQL Databases.

QUERY

Creating a collection:

db.createCollection("c1")

```
> db.createCollection("c1")
{ "ok" : 1 }
> show collections
c1
```

Inserting into a collection:

db.c1.insert({title:'blog1', body:'blog1 content',tags: ['blog','sports','arts'] })

db.c1.insert({title:'blog2',body:'blog2 content',tags: ['blog','news','information']})

Reading the collection:

db.c1.find()

```
> db.c1.find()
{ "_id" : ObjectId("62535ade7ea9dd3ac0a01ec3"), "title" : "blog1", "body" : "blog1 content", "tags" : [ "blog", "sports", "arts" ] }
{ "_id" : ObjectId("62535d8d7ea9dd3ac0a01ec4"), "title" : "blog2", "body" : "blog2 content", "tags" : [ "blog", "news", "information" ] }
```

db.c1.find().pretty()

```
> db.c1.find().pretty()
{
        "_id" : ObjectId("62535ade7ea9dd3ac0a01ec3"),
        "title" : "blog1",
        "body" : "blog1 content",
        "tags" : [
                "blog",
                "sports",
                "arts"
        ]
}
{
        "_id" : ObjectId("62535d8d7ea9dd3ac0a01ec4"),
        "title" : "blog2",
        "body" : "blog2 content",
        "tags" : [
                "blog",
                "news",
                "information"
        ]
}
>
```

Updating the Document:

db.c1.update({title:'blog1'},{"title": "blog3", "body": "blog3 content",

```
> db.c1.update({title:'blog1'},{"title": "blog3", "body": "blog3 content", "tags":["blog","educational","science"]})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.c1.find()
{ "_id" : ObjectId("62535ade7ea9dd3ac0a01ec3"), "title" : "blog3", "body" : "blog3 content", "tags" : [ "blog", "educational", "science" ] }
{ "_id" : ObjectId("62535d8d7ea9dd3ac0a01ec4"), "title" : "blog2", "body" : "blog2 content", "tags" : [ "blog", "news", "information" ] }
```

"tags":["blog","educational","science"]})

Deleting an item: db.c1.remove({title:'blog2'})

```
> db.c1.remove({title:'blog2'})
WriteResult({ "nRemoved" : 1 })
> db.c1.find()
{ "_id" : ObjectId("62535ade7ea9dd3ac0a01ec3"), "title" : "blog3", "body" : "blog3 content", "tags" : [ "blog", "educational", "science" ] }
```

Deleting a collection:

db.c1.drop()

```
> db.c1.drop()
true
> db.c1.find()
>
```

RESULT

PL/SQL program executed and the output was obtained, thus CO5 achieved

Date:_____

PROJECT- LIBRARY MANAGEMENT SYSTEM

This LibraryManagementSystem is a Java-based application with a graphical user interface (GUI) built using Swing. It offers functionalities for both Admin and Librarian roles in managing library resources.

Admins can add and remove both admins and librarians, while librarians can add and remove members and manage book inventory, including adding new books and updating borrowed items. The system interacts with a database using SQL commands for managing data, such as member details, book inventory, and borrowed books. It also includes methods to establish and close database connections, providing essential feedback through dialog boxes for successful or failed operations. The design is user-friendly, making library management tasks more efficient and organized.

The application's modular design with separate methods for each task improves readability and ease of maintenance.

- **Import Statements and Class Setup**

```
import java.awt.*;
import java.sql.*;
import javax.swing.*
public class LibraryManagementSystem extends JFrame {
        private Connection connection;

        public LibraryManagementSystem() {
                setTitle("Library Management System");
                setSize(400, 300);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
                setLocationRelativeTo(null);
                createMainMenu();
            }
        }
```

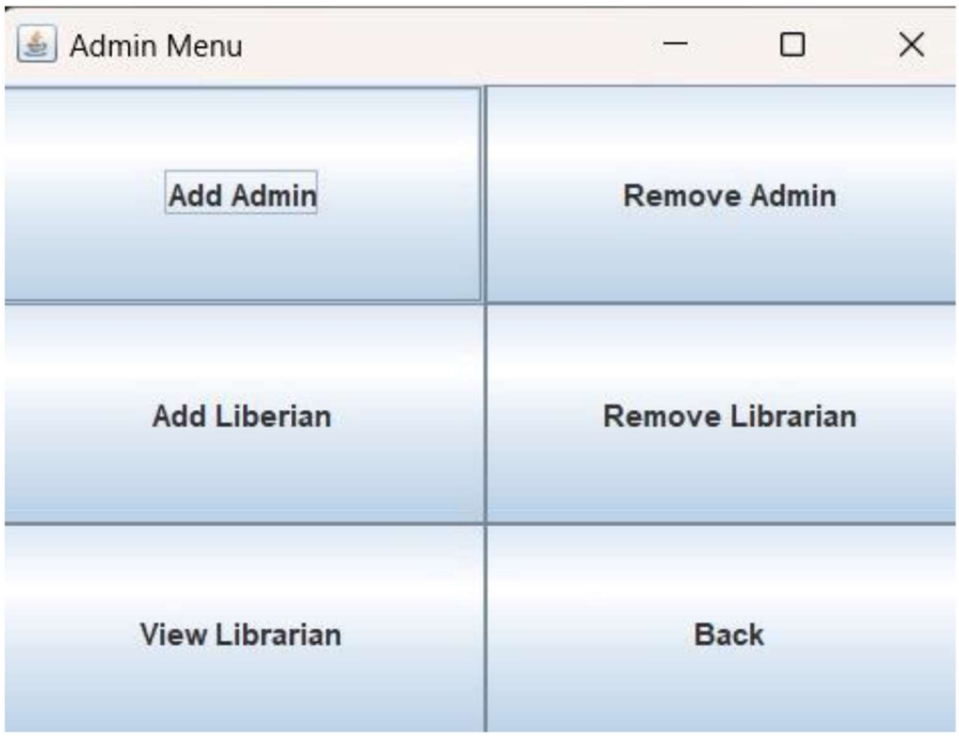
- **Main Menu Creation**

```
private void createMainMenu() {
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 1));

        JButton adminButton = new JButton("Admin Menu");
        JButton librarianButton = new JButton("Librarian Menu");
        JButton exitButton = new JButton("Exit");

        adminButton.addActionListener(e -> createAdminMenu());
        librarianButton.addActionListener(e -> createLibrarianMenu());
        exitButton.addActionListener(e -> System.exit(0));

        panel.add(adminButton);
        panel.add(librarianButton);
        panel.add(exitButton);

        add(panel);
        setVisible(true)
}
```

- **Admin Menu Creation**

```
private void createAdminMenu() {
        JFrame adminFrame = new JFrame("Admin Menu");
        adminFrame.setSize(400, 300);
        adminFrame.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        adminFrame.setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 1));
        JButton addAdminButton = new JButton("Add Admin");
        JButton removeAdminButton = new JButton("Remove Admin");
        JButton addLiberianButton = new JButton("Add Librarian");
        JButton removeLiberianButton = new JButton("Remove Librarian");
        JButton viewLiberianButton = new JButton("View Librarian");
        JButton backButton = new JButton("Back");

        addAdminButton.addActionListener(e -> addAdmin());
```

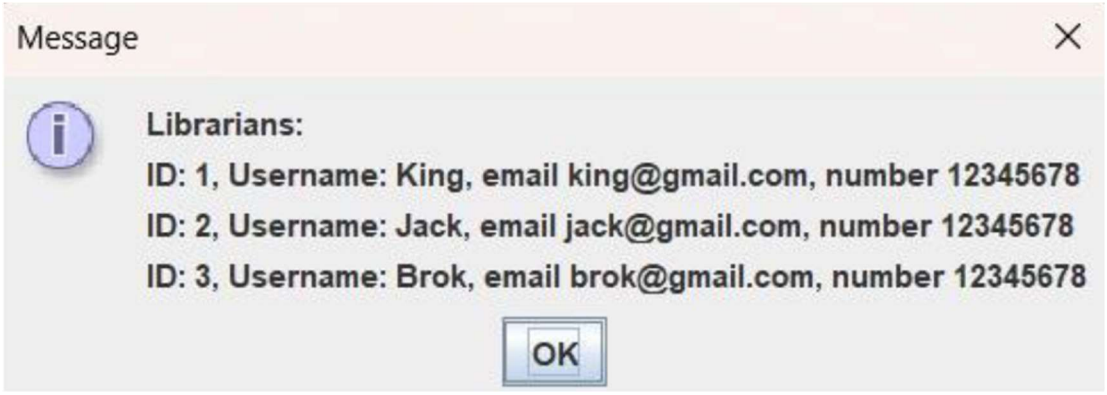
```
        removeAdminButton.addActionListener(e -> removeAdmin());
        addLiberianButton.addActionListener(e -> addLibrarian());
        removeLiberianButton.addActionListener(e -> removeLibrarian());
        viewLiberianButton.addActionListener(e -> displayLibrarians());
        backButton.addActionListener(e -> adminFrame.dispose());

        panel.add(addAdminButton);
        panel.add(removeAdminButton);
        panel.add(addLiberianButton);
        panel.add(removeLiberianButton);
        panel.add(viewLiberianButton);
        panel.add(backButton);

        adminFrame.add(panel);
        adminFrame.setVisible(true);
    }
```

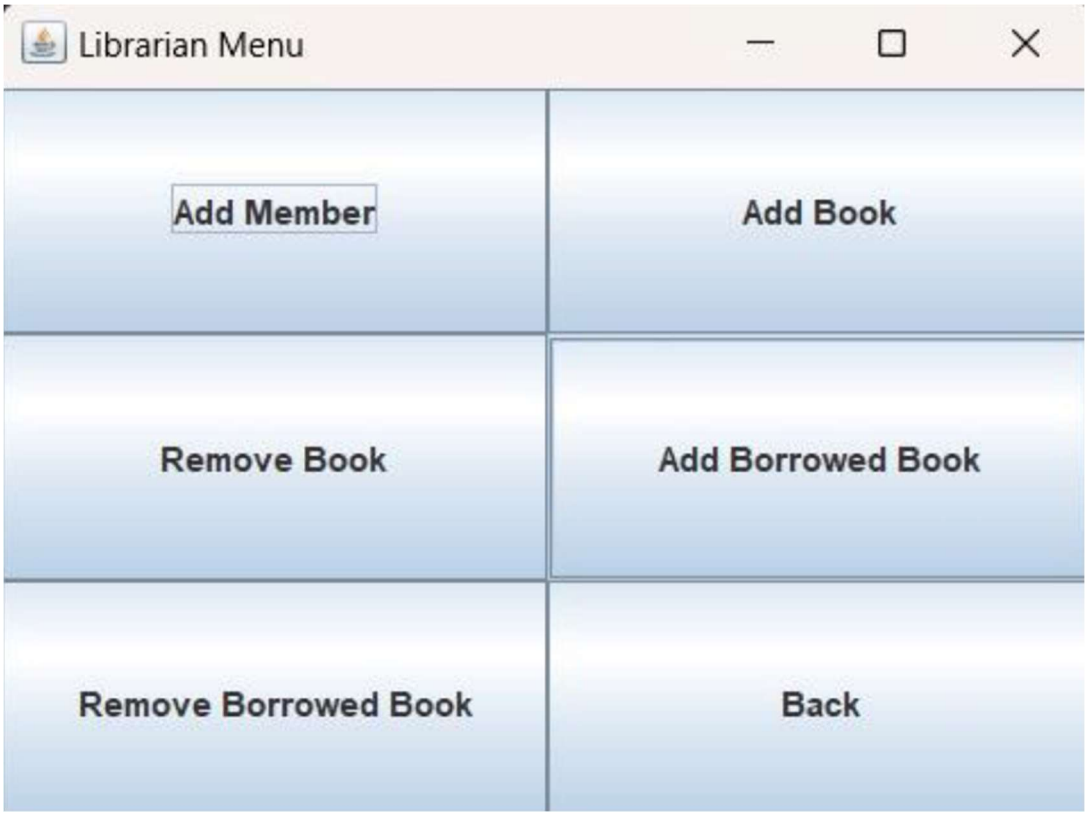


- **Librarian Menu Creation**

```
private void createLibrarianMenu() {
        JFrame librarianFrame = new JFrame("Librarian Menu");
        librarianFrame.setSize(400, 300);
        librarianFrame.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        librarianFrame.setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 1));

        JButton addMemberButton = new JButton("Add Member");
        JButton addBookButton = new JButton("Add Book");
        JButton removeBookButton = new JButton("Remove Book");
        JButton addBBookButton = new JButton("Add Borrowed Book");
        JButton removeBBookButton = new JButton("Remove Borrowed Book");
        JButton backButton = new JButton("Back");

        addMemberButton.addActionListener(e -> addMember());
        addBookButton.addActionListener(e -> addBook());
        removeBookButton.addActionListener(e -> removeBook());
        addBBookButton.addActionListener(e -> addBorrowedBook());
        removeBBookButton.addActionListener(e -> removeBorrowedBook());
        backButton.addActionListener(e -> librarianFrame.dispose());

        panel.add(addMemberButton);
        panel.add(addBookButton);
        panel.add(removeBookButton);
        panel.add(addBBookButton);
```

```
        panel.add(removeBBookButton);
        panel.add(backButton);
        librarianFrame.add(panel);
        librarianFrame.setVisible(true);
    }
```





- **Adding and Removing Admins**

```
private void addAdmin() {
        JTextField idField = new JTextField();
```

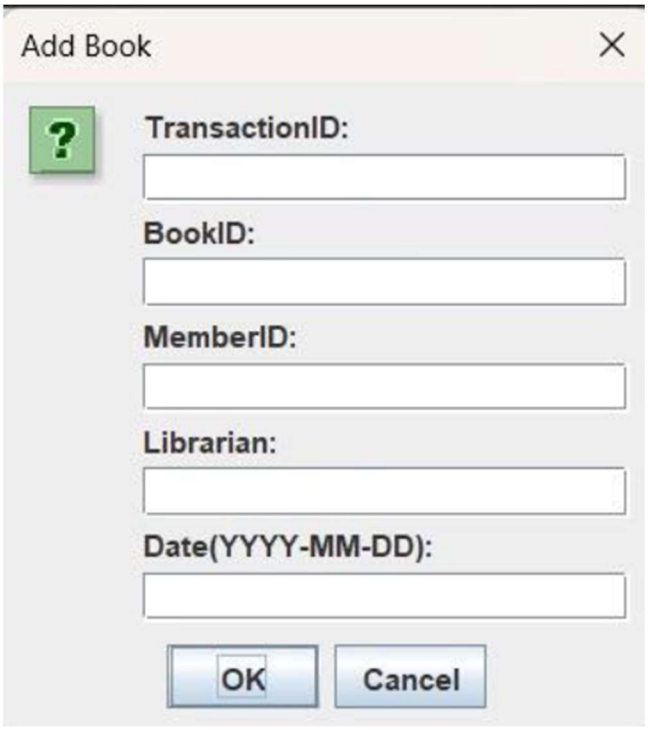
```
        JTextField nameField = new JTextField();
        JTextField emailField = new JTextField();
        JTextField phoneField = new JTextField();

        Object[] message = {
        "ID", idField,
        "Name:", nameField,
        "Email:", emailField,
        "Phone:", phoneField
        };
        int option = JOptionPane.showConfirmDialog(null, message, "Add Admin",
        JOptionPane.OK_CANCEL_OPTION);
        if (option == JOptionPane.OK_OPTION) {
                String id = idField.getText();
                String name = nameField.getText();
                String email = emailField.getText();
                String phone = phoneField.getText();
                executeUpdate("INSERT INTO Admins (admin_id, name, email, phone)
        VALUES ( ?, ?, ?, ?)", id, name, email, phone);
        }
    }
    private void removeAdmin() {
        String adminId = JOptionPane.showInputDialog("Enter Admin ID to remove:");
        if (adminId != null) {
          executeUpdate("DELETE FROM Admins WHERE admin_id = ?", adminId);
        }
    }
```

- **Adding and Removing Books and Members**

```
private void addMember() {
        JTextField idField = new JTextField();
        JTextField nameField = new JTextField();
        JTextField emailField = new JTextField();
        JTextField phoneField = new JTextField();

        Object[] message = {
                "ID", idField,
                "Name:", nameField,
                "Email:", emailField,
                "Phone:", phoneField
        };
        int option = JOptionPane.showConfirmDialog(null, message, "Add Member",
        JOptionPane.OK_CANCEL_OPTION);
        if (option == JOptionPane.OK_OPTION) {
                String id = idField.getText();
                String name = nameField.getText();
                String email = emailField.getText();
```

```
                String phone = phoneField.getText();
                executeUpdate("INSERT INTO Members (member_id, name, email,
                phone)
            VALUES ( ?, ?, ?, ?)", id, name, email, phone);
                }
            }
        private void addBook() {
                JTextField titleField = new JTextField();
                JTextField authorField = new JTextField();
                JTextField isbnField = new JTextField();
                JTextField yearField = new JTextField();
                JTextField copiesField = new JTextField();
                Object[] message = {
                    "Title:", titleField,
                    "Author:", authorField,
                    "ISBN:", isbnField,
                    "Published Year:", yearField,
                    "Total Copies:", copiesField
                };
                int option = JOptionPane.showConfirmDialog(null, message, "Add
                Book",
                JOptionPane.OK_CANCEL_OPTION);
                if (option == JOptionPane.OK_OPTION) {
                    String title = titleField.getText();
                    String author = authorField.getText();
                    String isbn = isbnField.getText();
                    String publishedYear = yearField.getText();
                    String totalCopies = copiesField.getText();
                executeUpdate("INSERT INTO Books (book_id, title, author, isbn,
```

```
                published_year, total_copies, available_copies) VALUES
                (Books_seq.NEXTVAL, ?,

                ?, ?, ?, ?, ?)", title, author, isbn, publishedYear, totalCopies,
                totalCopies);

                }

        }

        private void removeBook() {

                String bookId = JOptionPane.showInputDialog("Enter Book ID to
                remove:");

                if (bookId != null && !bookId.trim().isEmpty()) {

                executeUpdate("DELETE FROM Books WHERE book_id = ?",
                bookId);

                } else {

                JOptionPane.showMessageDialog(this, "Book ID cannot be empty.");

                }
}
```



- **Utility Methods**

```
private void executeUpdate(String query, String... params) {
  try (PreparedStatement stmt = connection.prepareStatement(query)) {
        for (int i = 0; i < params.length; i++) {
              stmt.setString(i + 1, params[i]);
}
int rowsAffected = stmt.executeUpdate();
if (rowsAffected > 0) {
        JOptionPane.showMessageDialog(this, "Operation successful.");
} else {
        JOptionPane.showMessageDialog(this, "Operation failed.");
}
} catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Database error: " +
        e.getMessage());
}
}
private void openConnection() {
        try {
              connection =
              DriverManager.getConnection("jdbc:oracle:thin:@localhost:15
              21:XE", "system",
              "password");
        } catch (SQLException e){
              JOptionPane.showMessageDialog(this, "Connection error: " +
              e.getMessage());
        }
        }
private void closeConnection() {
        try {
```

```
                    if (connection != null && !connection.isClosed()) {
                        connection.close();
                    }
                    } catch (SQLException e) {
                        JOptionPane.showMessageDialog(this, "Error closing
                        connection: " +
                        e.getMessage());
                    }
            }
        public static void main(String[] args) {
                    SwingUtilities.invokeLater(() -> new
                    LibraryManagementSystem().setVisible(true));
                    }
```

RESULT

Database for the library management system has been developed and values has been inserted and different operations have been executed successfully and the output has been obtained, thus CO6 achieved.