DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Periyar Nagar, Vallam Thanjavur - 613 403, Tamil Nadu, India
Phone: +91 - 4362 - 264600 Fax: +91- 4362 - 264660
Email: headcse@pmu.edu Web: www.pmu.edu

PERIYAR MANIAMMAI
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University)
Established Under Sec. 3 of UGC Act, 1956 • NAAC Accredited
think • innovate • transform

DATABASE MANAGEMENT SYSTEMS LABORATORY

III

Course Teacher

Department of Computer Science and Engineering

Vision

To produce professionals who can relate theory and practice, familiar with common themes and apply concepts of Computer Science and Engineering for Research and Societal development.

Mission

- To offer UG, PG, Ph.D. programme with state of art facilities in the field of Computer Science and Engineering.
- To prepare the students become globally competent by enhancing their skills to work in IT Industries and R & D organizations.
- To prepare the students with good ethical attitude and an ability to relate engineering issues to broader social context.
- To promote significant research in cutting edge Information and Communication technologies with environmental consciousness

Programme Educational Objectives

PEO1: Graduates will attain the expertise of analyzing and specifying the requirements for any computing system as well as capable of modeling, designing, implementing and verifying a computing system to meet specified requirements using contemporary tools.

PEO2: Graduates will possess diversified professional skills for successful career.

PEO3: Graduates of the programme will have the competencies for communicating, planning, coordinating, organizing, decision making and leading a team.

PEO4: Graduates of the programme will have knowledge of professional, interpersonal and ethical responsibility and will contribute to society through active research.

Programme Outcomes

PO1	an ability to apply knowledge of computing and mathematics appropriate to the discipline.
PO2	an ability to analyze a problem, interpret data, and define the computing system requirements which would be appropriate to the solution.
PO3	an ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
PO4	an ability to apply creativity in the design of systems which would help to investigate the complex problem and provide software solution.
PO5	an ability to use the computing techniques, skills, and modern system tools necessary for practice as a CSE professional
PO6	an ability to analyze the local and global impact of computing on individuals, organizations, and society
PO7	an ability to develop and use the software systems within realistic constraints environmental, health and safety, manufacturability, and sustainability considerations
PO8	an ability in an understanding of professional, ethical, legal, security and social issues and responsibilities
PO9	an ability to function effectively on teams and individually to accomplish a common goal
PO10	an ability to communicate effectively with a range of audiences by written and oral
PO11	ability to plan, organize and follow best practices and standards so that the project is completed as successfully by meeting performance, quality at CMM level, budget and time
PO12	an ability to engage in Lifelong learning and continuing professional development

Programmes Specific Outcomes

PSO1	ability to employ latest computer languages, environments and platforms for solving problems in the areas of emerging communication technologies.

PSO2 ability to use knowledge in data analytics and mining for industrial problems

Course Objectives and Course Outcomes

Course Objectives:

This course aims at

- facilitating the student to understand the various concepts and functionalities of Database Management Systems, the method and model to store data.
- How to manipulate through query languages, the effective designing of relational database .
- How the system manages the concurrent usage of data in multi user environment.

Course Outcomes

CO1	***Construct*** queries with relational database system with the basics of SQL
CO2	***Relate and Apply*** the design principles for logical design of databases, including ER model and normalization approach
CO3	***Define and Explain***the basic database storage structures and access techniques: file and page organizations, indexing methods including B-tree, B+ tree andhashing.
CO4	***Define and Explain the*** basic issues of transaction processing and concurrency control.
CO5	**Work** successfully in a team by design and development of database application systems.

LIST OF EXPERIMENTS

EX.NO.1 ENTITY RELATIONSHIP MODEL

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them.

ENTITY

An entity can be a real-world object, that can be easily identifiable. For example, in a school database, student, teacher, class, and course offered can be considered as entity. All these entities have some attributes or properties.

An entity set is a collection of similar type of entities. For example, Students may contain all the students of a school; likewise a Teachers may contain all the teachers of a school Entity sets need not be disjoint.

DESIGN AN ENTITY RELATIONSHIP (ER) MODEL FOR A UNIVERSITY DATABASE

1. An University contains many departments
2. Each department can offer any number of courses
3. Many instructors can work in a department
4. An instructor can work only in one department
5. For each department there is a Head
6. An instructor can be head of only one department
7. Each instructor can take any number of courses
8. A course can be taken by only one instructor
9. A student can enroll for any number of courses
10. Each course can have any number of students

STEP 1 : IDENTIFY THE ENTITIES

The entities are

1. Department
2. Course
3. Instructor
4. Student

STEP 2 : IDENTIFY THE RELATIONSHIPS

1. One department offers many courses. But one particular course can be offered by only one department. hence the cardinality between department and course is One to Many (1:N)
2. One department has multiple instructors. But instructor belongs to only one department. Hence the cardinality between department and instructor is One to Many (1:N)
3. One department has only one head and one head can be the head of only one department. Hence the cardinality is one to one. (1:1)
4. One course can be enrolled by many students and one student can enroll for many courses. Hence the cardinality between course and student is Many to Many (M:N)
5. One course is taught by only one instructor. But one instructor teaches many courses. Hence the cardinality between course and instructor is Many to One (N :1)

STEP 3: IDENTIFY THE PRIMARY KEY ATTRIBUTE

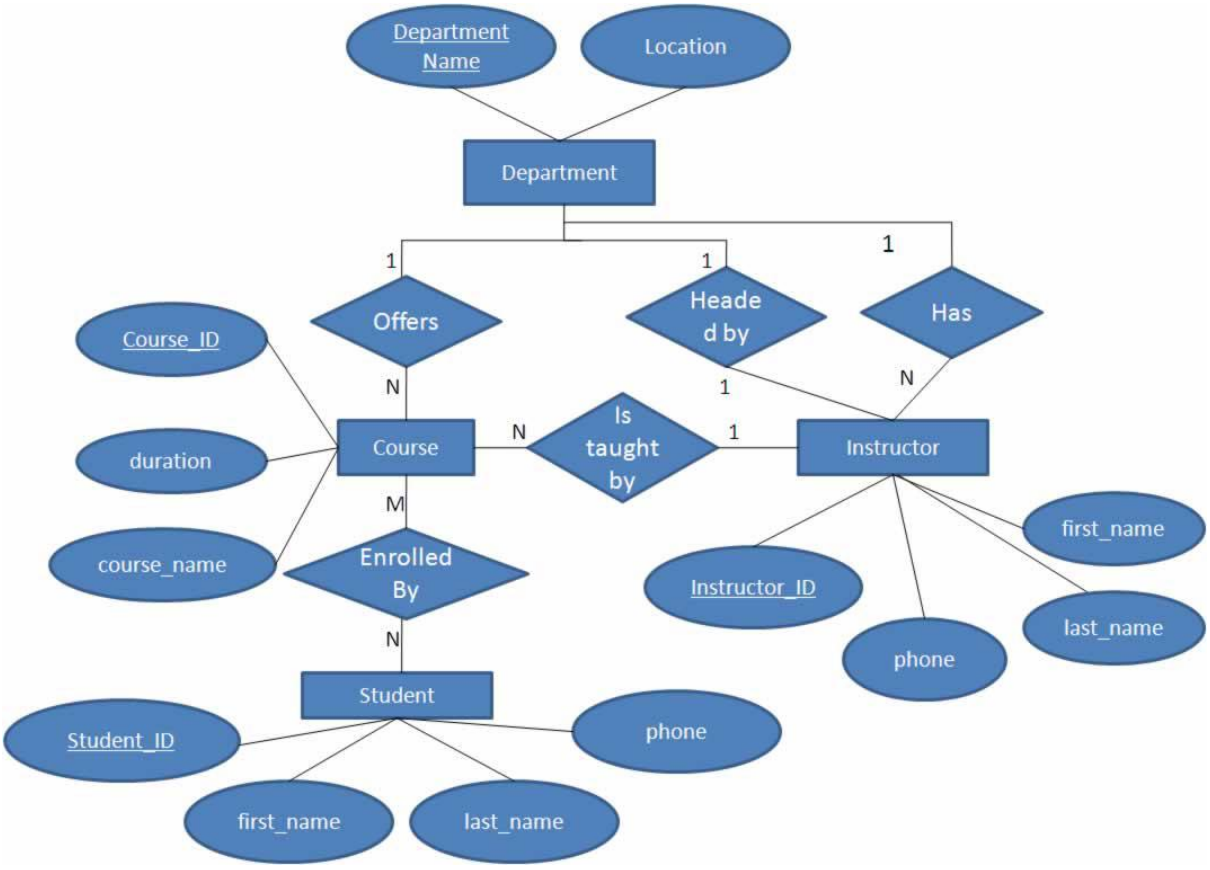
- "Department_Name" can identify a department uniquely. Hence Department_Name is the primary key attribute for the Entity "Department".
- Course_ID is the primary key attribute for "Course" Entity.
- Student_ID is the primary key attribute for "Student" Entity.
- Instructor_ID is the primary key attribute for "Instructor" Entity.

STEP 4: IDENTIFY OTHER RELEVANT ATTRIBUTES

- For the department entity, other attributes are location
- For course entity, other attributes are course_name,duration
- For instructor entity, other attributes are first_name, last_name, phone
- For student entity, first_name, last_name, phone

STEP 5: DRAW COMPLETE ER DIAGRAM

By connecting all these details, we can now draw ER diagram as given below.



EX.NO.2.1 DATA DEFINITION LANGUAGE (DDL) COMMANDS

The DDL provides commands for defining relation schemas, deleting relations and modifying relation schemas.

DDL is used to:

- Create an object
- Alter the structure of an object
- To drop the object created.

The commands used are:

1. Create
2. Alter
3. Drop
4. Truncate
5. Rename

1. CREATE COMMAND

This command is used to create a table

Syntax:

```
Create table tablename
(column_name1 data_ type,
column_name2 data_ type …)
```

2.ALTER COMMAND

This command is to add attributes to an existing relation

Syntax:

```
Alter table r add A D;
```

Where r -name of the existing relation,
A - Name of the attribute to be added
D-type of the added attribute

3. DROP COMMAND

Drop command deletes all information about the dropped relation from the database

Syntax:

```
DROP TABLE <Tablename>;
```

4.TRUNCATE COMMAND

Truncate command removes all the records from the table

Syntax:

```
TRUNCATE TABLE  <Table_name>
```

5. RENAME COMMAND

Rename command is used to rename the objects

```
RENAME  <oldTableName> To < NewTableName>
```

EXERCISES:

QUERY: 1
Write a query to create a table employee with employee number, employee name, designation and salary.

SQL>CREATE TABLE EMP (EMPNO NUMBER (4), ENAME VARCHAR2 (10), DESIGNATION VARCHAR2 (10), SALARY NUMBER (8,2));

OUTPUT:
Table created.

QUERY: 2
Write a query to display the column name and data type of the table employee.

Syntax :

DESC <TABLE NAME>;

SQL>DESC EMP;

OUTPUT:

Name Null?	Type
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2)

QUERY: 3
Write a query to create a table from an existing table with all the fields

SQL> CREATE TABLE EMP1 AS SELECT * FROM EMP;

OUTPUT:
Table created.

To view the EMP1 table

SQL> DESC EMP1

Name Null?	Type
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2)

QUERY: 4

Write a query to create a table from an existing table with selected fields

Syntax to Create a table from an existing table with selected fields

SQL> CREATE TABLE <TARGET TABLE NAME> SELECT EMPNO, ENAME
FROM <SOURCE TABLE NAME>;

SQL> CREATE TABLE EMP2 AS SELECT EMPNO, ENAME FROM EMP;

OUTPUT:

Table created.

To view EMP2

SQL> DESC EMP2

Name Null?	Type
EMPNO	NUMBER (4)
ENAME	VARCHAR2 (10)

ALTER & MODIFICATION ON TABLE

QUERY: 5

Write a Query to Alter the column EMPNO NUMBER (4) TO EMPNO NUMBER(6).

Syntax for Alter & Modify on a Single Column:

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME><DATATYPE> (SIZE);

SQL>ALTER TABLE EMP MODIFY EMPNO NUMBER (6);

OUTPUT:
Table altered.

To view EMP
SQL> DESC EMP;

Name Null?	Type
EMPNO	NUMBER(6)
ENAME	VARCHAR2(10)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2)

QUERY: 6
Write a Query to Alter the table employee with multiple columns (EMPNO,ENAME.)

Syntax for alter table with multiple column:
SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME1><DATATYPE> (SIZE), MODIFY <COLUMN NAME2><DATATYPE> (SIZE)……;

SQL>ALTER TABLE EMP MODIFY (EMPNO NUMBER (7), ENAME VARCHAR2(12);

OUTPUT:
Table altered.

SQL> DESC EMP;

Name Null?	Type
EMPNO	NUMBER(7)
ENAME	VARCHAR2(12)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2);

QUERY: 7
Write a query to add a new column in employee relation.

Syntax for add a new column:
SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME><DATA TYPE><SIZE>);

SQL> ALTER TABLE EMP ADD QUALIFICATION VARCHAR2(6);

OUTPUT:
Table altered.
SQL> DESC EMP;

Name Null?	Type
EMPNO	NUMBER(7)
ENAME	VARCHAR2(12)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2)
QUALIFICATION	VARCHAR2(6)

QUERY: 8
Write a query to add multiple columns in employee relation.

Syntax for add a new column:
SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME1><DATA TYPE><SIZE>,(<COLUMN NAME2><DATA TYPE><SIZE>, ……;

SQL>ALTER TABLE EMP ADD (DOB DATE, DOJ DATE);

OUTPUT:
Table altered.

SQL> DESC EMP;

Name Null?	Type
EMPNO	NUMBER(7)
ENAME	VARCHAR2(12)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2)
QUALIFICATION	VARCHAR2(6)
DOB	DATE
DOJ	DATE

DROP COMMAND

QUERY: 9
Write a query to drop the table

SQL> DROP TABLE STUDENT;

OUTPUT:
Table dropped

QUERY: 10
Write a query to drop a column from an existing table employee

Syntax to drop a column in the existing table:
SQL> ALTER TABLE <TABLE NAME> DROP COLUMN <COLUMN NAME>;

SQL> ALTER TABLE EMP DROP COLUMN DOJ;

OUTPUT:
Table altered.

SQL> DESC EMP;

Name Null?	Type
EMPNO	NUMBER(7)
ENAME	VARCHAR2(12)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2)
QUALIFICATION	VARCHAR2(6)
DOB	DATE

QUERY: 11
Write a query to drop multiple columns from employee

Syntax for add a new column:
SQL> ALTER TABLE <TABLE NAME> DROP <COLUMNNAME1>,<COLUMN NAME2>,…… ;

SQL> ALTER TABLE EMP DROP (DOB, QUALIFICATION);

OUTPUT:
Table altered.

SQL> DESC EMP;

Name Null?	Type
EMPNO	NUMBER(7)
ENAME	VARCHAR2(12)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2)

RENAME COMMAND:

QUERY: 12
Write a query to rename table emp to employee

Syntax to rename the table name:

RENAME <oldTableName> To < NewTableName>

SQL> RENAME EMP TO EMPLOYEE;

SQL> DESC EMPLOYEE;

Name Null?	Type
EMPNO	NUMBER(7)
ENAME	VARCHAR2(12)
DESIGNATION	VARCHAR2(10)
SALARY	NUMBER(8,2)

EX.NO.2.2	CONSTRAINTS

Aim:
To execute constraints in SQL

Integrity Constraints are part of the table definition that limits and restrict the value entered into its columns. Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency.

TYPES OF CONSTRAINTS:
1) Primary key
2) Foreign key/references
3) Check
4) Unique
5) Not null

PRIMARY KEY

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary key must contain unique values. This column cannot have NULL values.

To create a PRIMARY KEY in Customer table

CREATE TABLE CUSTOMERS(ID INT(5), NAME VARCHAR (20), AGE INT(3) ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID));

To create a PRIMARY KEY constraint on the "ID" column when CUSTOMERS table already exists, following SQL syntax is used.

ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);

UNIQUE Constraint

The UNIQUE Constraint prevents two records from having identical values in a particular column.

The following query creates a new table called CUSTOMER. Here ID column is set to UNIQUE, so that two records cannot have same ID.

CREATE TABLE CUSTOMER (
ID INT NOT NULL UNIQUE,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
);

If CUSTOMER table has already been created, then to add a UNIQUE constraint to ID column,

ALTER TABLE CUSTOMER MODIFY ID INT NOT NULL UNIQUE;

FOREIGN KEY

To ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This condition is called referential integrity. Foreign Key can be specified as part of the SQL, create table statement by using the foreign key clause.

Consider the structure of the two tables as follows:

CUSTOMERS table:

CREATE TABLE CUSTOMERS (
ID INT NOT NULL,
NAMEVARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID));

ORDERS table:

CREATE TABLE ORDERS (
ORD_ID INT NOT NULL,
ORD_DATE DATE,
FOREIGN KEY ID references CUSTOMERS);

CHECK Constraint

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and doesn't entered into the table.

For example, the following query creates a new table called CUSTOMERS. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER below 18 years:

```
CREATE TABLE CUSTOMERS(
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL CHECK (AGE >= 18),
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID));
```

If CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, the following statement can be given.

```
ALTER TABLE CUSTOMERS MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );
```

NOT NULL

The not null constraint prohibits the insertion of a null value for the attribute

Syntax:

```
Name  varchar(20) not null
Budget varchar(12,2) not null
```

For example, the following query creates a new table called CUSTOMERS. The NAME and AGE attribute is declared as NOT NULL.

```
CREATE TABLE CUSTOMERS (
ID INT,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL CHECK (AGE >= 18),
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID));
```

EX.NO.3 DATA MANIPULATION LANGUANGE(DML) COMMANDS

DML (DATA MANIPULATION LANGUAGE)

- SELECT
- INSERT
- DELETE
- UPDATE

1. INSERT
INSERT command is used to INSERT object in the database.

2. SELECT
SELECT command is used to SELECT the object from the database.

3. UPDATE
UPDATE command is used to UPDATE the records from the table

4.DELETE
DELETE command is used to DELETE the Records form the table

INSERT

QUERY: 1
Write a query to insert the records in employee relation

Syntax
SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',.....);

INSERT A RECORD FROM AN EXISTING TABLE:

SQL>INSERT INTO EMP VALUES(101,'NAGARAJ,'LECTURER',15000);
1 row created.

-

SELECT

QUERY: 2
Write a query to display the records from employee relation.

Syntax

SQL> SELECT * FROM <TABLE NAME>;

SQL> SELECT * FROM EMP;

EMPNO ENAME DESIGNATION SALARY
---------- ---------- ---------- ----------
101 NAGARAJAN LECTURER 15000

INSERT A RECORD USING SUBSITUTION METHOD

QUERY: 3
Write a query to insert the records in to employee using substitution method.

Syntax:

SQL :> INSERT INTO <TABLE NAME> VALUES< '&column name', '&column name 2',.....);

SQL> INSERT INTO EMP VALUES (&EMPNO,'&ENAME','&DESIGNATION','&SALARY');
Enter value for empno: 102
Enter value for ename: SARAVANAN
Enter value for designatin: LECTURER
Enter value for salary: 15000

old 1: INSERT INTO EMP
VALUES(&EMPNO,'&ENAME','&DESIGNATION','&SALARY')
new 1: INSERT INTO EMP VALUES(102,'SARAVANAN','LECTURER','15000')
1 row created.
SQL> /
Enter value for empno: 103
Enter value for ename: PANNERSELVAM
Enter value for designatin: ASST. PROF
Enter value for salary: 20000
old 1: INSERT INTO EMP
VALUES(&EMPNO,'&ENAME','&DESIGNATION','&SALARY')
new 1: INSERT INTO EMP VALUES(103,'PANNERSELVAM','ASST.
PROF','20000')
1 row created.
SQL> /
Enter value for empno: 104
Enter value for ename: CHARULATHA
Enter value for designatin: HOD, PROF
Enter value for salary: 45000
old 1: INSERT INTO EMP
VALUES(&EMPNO,'&ENAME','&DESIGNATION','&SALARY')

new 1: INSERT INTO EMP VALUES(104,'CHARULATHA','HOD, PROF','45000')
1 row created.

SQL> SELECT * FROM EMP;
EMPNO ENAME DESIGNATION SALARY
---------- ---------- ---------- ----------
101 NAGARAJAN LECTURER 15000
102 SARAVANAN LECTURER 15000
103 PANNERSELVAM ASST. PROF 20000
104 CHARULATHA HOD, PROF 45000

UPDATE

QUERY: 4
Write a query to update the records in employee relation.
Syntax
SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

SQL> UPDATE EMP SET SALARY=16000 WHERE EMPNO=101;
1 row updated.

SQL> SELECT * FROM EMP;

EMPNO ENAME DESIGNATION SALARY
---------- ---------- ---------- ----------
101 NAGARAJAN LECTURER 16000
102 SARAVANAN LECTURER 15000
103 PANNERSELVAM ASST. PROF 20000
104 CHARULATHA HOD, PROF 45000

UPDATE MULTIPLE COLUMNS
QUERY: 5
Q5. Write a query to update multiple records from employee.
Syntax
SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

SQL>UPDATE EMP SET SALARY = 16000, DESIGNATION='ASST. PROF' WHERE EMPNO=102;
1 row updated.

SQL> SELECT * FROM EMP;

EMPNO ENAME DESIGNATION SALARY
---------- ---------- ---------- ----------

101 NAGARAJAN LECTURER 16000
102 SARAVANAN ASST. PROF 16000
103 PANNERSELVAM ASST. PROF 20000
104 CHARULATHA HOD, PROF 45000

DELETE

QUERY: 6

Write a query to delete records from employee relation.

Syntax

SQL> DELETE <TABLE NAME> WHERE <COLUMN NAME>=<VALUE>;

SQL> DELETE EMP WHERE EMPNO=103;

1 row deleted.

SQL> SELECT * FROM EMP;

EMPNO ENAME DESIGNATION SALARY
---------- ---------- ---------- ----------
101 NAGARAJAN LECTURER 16000
102 SARAVANAN ASST. PROF 16000
104 CHARULATHA HOD, PROF 45000

EX.NO.3 DCL (DATA CONTROL LANGUAGE) COMMANDS

CREATING A USER

SQL>CONNECT SYSTEM/MANAGER;
SQL>CREATE USER "USERNAME" IDENTIFIED BY "PASSWORD"
SQL>GRANT DBA TO "USERNAME"
SQL>CONNECT "USERNAME"/"PASSWORD";

EXAMPLE

CREATING A USER
SQL>CONNECT SYSTEM/MANAGER;
SQL>CREATE USER CSE2 IDENTIFIED BY CSECSE;
SQL>GRANT DBA TO CSE2;
SQL>CONNECT CSE2/CSECSE;
SQL>REVOKE DBA FROM CSE2;

EX.NO.4 NESTED SUBQUERIES

Aim:
To execute nested queries in SQL.

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

SYNTAX FOR CREATING A TABLE:

SQL: CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME.1 <DATATYPE> (SIZE) ……);

```
SQL> CREATE TABLE EMP2(EMPNO NUMBER(5),
ENAME VARCHAR2(20),
JOB VARCHAR2(20),
SAL NUMBER(6),
MGRNO NUMBER(4),
DEPTNO NUMBER(3));
```

SYNTAX FOR INSERTING RECORDS INTO A TABLE:
SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',…..);

INSERTION

```
SQL> INSERT INTO EMP2 VALUES(1001,'MAHESH','PROGRAMMER',15000,1560,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1002,'MANOJ','TESTER',12000,1560,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1003,'KARTHIK','PROGRAMMER',13000,1400,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1004,'NARESH','CLERK',1400,1400,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1005,'MANI','TESTER',13000,1400,200);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1006,'VIKI','DESIGNER',12500,1560,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1007,'MOHAN','DESIGNER',14000,1560,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1008,'NAVEEN','CREATION',20000,1400,201);
1 ROW CREATED.
SQL> INSERT INTO EMP2 VALUES(1009,'PRASAD','DIR',20000,1560,202);
1 ROW CREATED.
```

SQL> INSERT INTO EMP2 VALUES(1010,'AGNESH','DIR',15000,1400,200);
1 ROW CREATED.

SYNTAX FOR SELECTING RECORDS FROM THE TABLE:
SQL> SELECT * FROM <TABLE NAME>;
SQL> SELECT *FROM EMP2;
EMPNO ENAME JOB SAL MGRNO DPTNO
---------- ---------- ---------- ---------- ---------- ----------
1001 MAHESH PROGRAMMER 15000 1560 200
1002 MANOJ TESTER 12000 1560 200
1003 KARTHIK PROGRAMMER 13000 1400 201
1004 NARESH CLERK 1400 1400 201
1005 MANI TESTER 13000 1400 200
1006 VIKI DESIGNER 12500 1560 201
1007 MOHAN DESIGNER 14000 1560 201
1008 NAVEEN CREATION 20000 1400 201
1009 PRASAD DIR 20000 1560 202
1010 AGNESH DIR 15000 1400 200

TABLE- 2
SYNTAX FOR CREATING A TABLE:
SQL: CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE), COLUMN NAME.1 <DATATYPE> (SIZE) ……);
SQL> CREATE TABLE DEPT2(DEPTNO NUMBER(3),
DEPTNAME VARCHAR2(10),
LOCATION VARCHAR2(15));
Table created.

SYNTAX FOR INSERT RECORDS IN TO A TABLE:
SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',…..);
INSERTION
SQL> INSERT INTO DEPT2 VALUES(107,'DEVELOP','ADYAR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(201,'DEBUG','UK');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(200,'TEST','US');
SQL> INSERT INTO DEPT2 VALUES(201,'TEST','USSR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(108,'DEBUG','ADYAR');
1 ROW CREATED.
SQL> INSERT INTO DEPT2 VALUES(109,'BUILD','POTHERI');
1 ROW CREATED.

SYNTAX FOR SELECT RECORDS FROM THE TABLE:

SQL> SELECT * FROM <TABLE NAME>;

SQL> SELECT *FROM DEPT2;

DEPTNO DEPTNAME LOCATION

---------- ---------- ---------------

107 DEVELOP ADYAR

201 DEBUG UK

200 TEST US

201 TEST USSR

108 DEBUG ADYAR

109 BUILD POTHERI

6 rows selected.

GENERAL SYNTAX FOR NESTED QUERY:

SELECT "COLUMN_NAME1"

FROM "TABLE_NAME1"

WHERE "COLUMN_NAME2" [COMPARISON OPERATOR]

(SELECT "COLUMN_NAME3"

FROM "TABLE_NAME2"

WHERE [CONDITION])

SYNTAX NESTED QUERY STATEMENT:

SQL> SELECT <COLUMN_NAME> FROM FRORM <TABLE _1> WHERE

<COLUMN_NAME><RELATIONAL _OPERATION> 'VALUE'

(SELECT (AGGRECATE FUNCTION) FROM <TABLE_1> WHERE <COLUMN

NAME> = 'VALUE'

(SELECT <COLUMN_NAME> FROM <TABLE_2> WHERE <COLUMN_NAME=

'VALUE'));

NESTED QUERY STATEMENT:

SQL> SELECT ENAME FROM EMP2 WHERE SAL>

(SELECT MIN(SAL) FROM EMP2 WHERE DPTNO=

(SELECT DEPTNO FROM DEPT2 WHERE LOCATION='UK'));

Nested Query Output:

ENAME

MAHESH

MANOJ

KARTHIK

MANI

VIKI

MOHAN

NAVEEN

PRASAD

EX.NO.5 CURSOR

Aim:
To execute explicit and implicit cursor in PL/SQL.

CREATE A TABLE

SQL> select * from EMP;

EMPNO ENAME JOB MGR HIREDATE SAL COMM
---------- ---------- --------- ---------- --------- ---------- ----------
DEPTNO

7369 SMITH CLERK 7902 17-DEC-80 800
20
7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300
30
7521 WARD SALESMAN 7698 22-FEB-81 1250 500
30

EMPNO ENAME JOB MGR HIREDATE SAL COMM
---------- ---------- --------- ---------- --------- ---------- ----------
DEPTNO

7566 JONES MANAGER 7839 02-APR-81 2975
20
7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400
30
7698 BLAKE MANAGER 7839 01-MAY-81 2850
30

EMPNO ENAME JOB MGR HIREDATE SAL COMM
---------- ---------- --------- ---------- --------- ---------- ----------
DEPTNO

7782 CLARK MANAGER 7839 09-JUN-81 2450
10
7788 SCOTT ANALYST 7566 09-DEC-82 3000
20
7839 KING PRESIDENT 17-NOV-81 5000
10

EMPNO ENAME JOB MGR HIREDATE SAL COMM
---------- ---------- --------- ---------- --------- ---------- ----------
DEPTNO

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0
30
7876 ADAMS CLERK 7788 12-JAN-83 1100
20
7900 JAMES CLERK 7698 03-DEC-81 950
30

EMPNO ENAME JOB MGR HIREDATE SAL COMM
---------- ---------- --------- ---------- --------- ---------- ----------
DEPTNO

7902 FORD ANALYST 7566 03-DEC-81 3000
20
7934 MILLER CLERK 7782 23-JAN-82 1300
10
14 rows selected.

IMPLICIT CURSOR:

```
SQL> DECLARE
2 ena EMP.ENAME%TYPE;
3 esa EMP.SAL%TYPE;
4 BEGIN
5 SELECT ENAME,SAL INTO ENA,ESA FROM EMP
6 WHERE EMPNO = &EMPNO;
7 DBMS_OUTPUT.PUT_LINE('NAME :' || ENA);
8 DBMS_OUTPUT.PUT_LINE('SALARY :' || ESA);
9 EXCEPTION
10 WHEN NO_DATA_FOUND THEN
11 DBMS_OUTPUT.PUT_LINE('Employee no does not exits');
12 END;
13 /
```

OUTPUT:

Enter value for empno: 7844
old 6: WHERE EMPNO = &EMPNO;
new 6: WHERE EMPNO = 7844;
PL/SQL procedure successfully completed.

EXPLICIT CURSORS:

```
SQL> DECLARE
2 ena EMP.ENAME%TYPE;
3 esa EMP.SAL%TYPE;
4 CURSOR c1 IS SELECT ename,sal FROM EMP;
5 BEGIN
6 OPEN c1;
7 FETCH c1 INTO ena,esa;
8 DBMS_OUTPUT.PUT_LINE(ena || ' salary is $ ' || esa);
9 FETCH c1 INTO ena,esa;
```

```
10 DBMS_OUTPUT.PUT_LINE(ena || ' salary is $ ' || esa);
11 FETCH c1 INTO ena,esa;
12 DBMS_OUTPUT.PUT_LINE(ena || ' salary is $ ' || esa);
13 CLOSE c1;
14 END;
15 /
```

OUTPUT:

```
SMITH salary is $ 800
ALLEN salary is $ 1600
WARD salary is $ 1250
```

EX.NO.6 **TRIGGER**

AIM

To execute trigger for before update, after update, delete, insert operations on a table.

SQL> create table emp(id number(3),name varchar2(10),income number(4),expense number(3),savings number(3));
Table created.

SQL> insert into emp values(2,'kumar',2500,150,650);
1 row created.
SQL> insert into emp values(3,'venky',5000,900,950);
1 row created.
SQL> insert into emp values(4,'anish',9999,999,999);
1 row created.

SQL> select * from emp ;
ID NAME INCOME EXPENSE SAVINGS
---------- ---------- ---------- ---------- ----------
2 kumar 2500 150 650
3 venky 5000 900 950
4 anish 9999 999 999

PROGRAM- TRIGGER AFTER UPDATE

```
SQL> CREATE OR REPLACE TRIGGER t-check
AFTER UPDATE OR INSERT OR DELETE ON EMP
FOR EACH ROW
BEGIN
IF UPDATING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS UPDATED');
ELSIF INSERTING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS INSERTED');
ELSIF DELETING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS DELETED');
END IF;
END;
/
Trigger created.
SQL> update emp set income =900 where empname='kumar';
TABLE IS UPDATED
1 row updated.
SQL> insert into emp values ( 4,'Chandru',700,250,80);
TABLE IS INSERTED
1 row created.
SQL> DELETE FROM EMP WHERE EMPID = 4;
```

```
TABLE IS DELETED
1 row deleted.
SQL> select * from emp;
EMPID EMPNAME INCOME EXPENSE SAVINGS
--------- ---------- ---------- ---------- ----------
2 vivek 830 150 100
3 kumar 5000 550 50
9 vasanth 987 6554 644
```

PROGRAM - TRIGGER BEFORE UPDATE

```
-----------------------------------------------------
SQL> CREATE OR REPLACE TRIGGER  EMP1
BEFORE UPDATE OR INSERT OR DELETE ON EMPLOYEE
FOR EACH ROW
BEGIN
IF UPDATING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS UPDATED');
ELSIF INSERTING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS INSERTED');
ELSIF DELETING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS DELETED');
END IF;
END;
/
Trigger created.
SQL> INSERT INTO EMP VALUES (4,'SANKAR',700,98,564);
TABLE IS INSERTED
1 row created.
SQL> UPDATE EMP SET EMPID = 5 WHERE EMPNAME = 'SANKAR';
TABLE IS UPDATED
1 row updated.
SQL> DELETE EMP WHERE EMPNAME='SANKAR';
TABLE IS DELETED
1 row deleted.
```

EX.NO.7	VIEWS

AIM

To create and execute views in SQL.

COMMAND : **CREATE VIEW**

COMMAND DESCRIPTION: **CREATE VIEW** command is used to define a view.

CREATION OF TABLE

```
SQL> CREATE TABLE EMPLOYEE (
EMPLOYEE_NAMEVARCHAR2(10),
EMPLOYEE_NONUMBER(8),
DEPT_NAME VARCHAR2(10),
DEPT_NO NUMBER (5),DATE_OF_JOIN DATE);
Table created.
```

TABLE DESCRIPTION

```
SQL> DESC EMPLOYEE;

NAME NULL? TYPE
------------------------------ -------- ------------------------
EMPLOYEE_NAME VARCHAR2(10)
EMPLOYEE_NO NUMBER(8)
DEPT_NAME VARCHAR2(10)
DEPT_NO NUMBER(5)
DATE_OF_JOIN DATE
```

SYNTAX FOR CREATION OF VIEW

```
SQL> CREATE <VIEW><VIEW NAME> AS SELECT
<COLUMN_NAME_1>, <COLUMN_NAME_2> FROM <TABLE NAME>;
```

CREATION OF VIEW

```
SQL> CREATE VIEW EMPVIEW AS SELECT
EMPLOYEE_NAME,EMPLOYEE_NO,DEPT_NAME,DEPT_NO,DATE_OF_JOIN FROM
EMPLOYEE;
VIEW CREATED.
```

DESCRIPTION OF VIEW

```
SQL> DESC EMPVIEW;

NAME NULL? TYPE
----------------------------------------- -------- ----------------------------
EMPLOYEE_NAME VARCHAR2(10)
EMPLOYEE_NO NUMBER(8)
DEPT_NAME VARCHAR2(10)
DEPT_NO NUMBER(5)
```

DISPLAY VIEW:

```
SQL> SELECT * FROM EMPVIEW;
EMPLOYEE_N EMPLOYEE_NO DEPT_NAME DEPT_NO
---------- ---------- ---------- ----------
RAVI 124 ECE 89
VIJAY 345 CSE 21
RAJ 98 IT 22
GIRI 100 CSE 67
```

INSERTION INTO VIEW

INSERT STATEMENT:

SYNTAX:

```
SQL> INSERT INTO <VIEW_NAME> (COLUMN NAME1,…….)
VALUES(VALUE1,….);
SQL> INSERT INTO EMPVIEW VALUES ('SRI', 120,'CSE', 67,'16-NOV-1981');
1 ROW CREATED.

SQL> SELECT * FROM EMPVIEW;
EMPLOYEE_N EMPLOYEE_NO DEPT_NAME DEPT_NO
---------- ---------- ---------- ----------
RAVI 124 ECE 89
VIJAY 345 CSE 21
RAJ 98 IT 22
GIRI 100 CSE 67
SRI 120 CSE 67

SQL> SELECT * FROM EMPLOYEE;
EMPLOYEE_N EMPLOYEE_NO DEPT_NAME DEPT_NO DATE_OF_J
---------- ---------- ---------- ---------- ---------
RAVI 124 ECE 89 15-JUN-05
VIJAY 345 CSE 21 21-JUN-06
RAJ 98 IT 22 30-SEP-06
```

GIRI 100 CSE 67 14-NOV-81
SRI 120 CSE 67 16-NOV-81

DELETION OF VIEW:
DELETE STATEMENT:
SYNTAX:
SQL> DELETE <VIEW_NAME>WHERE <COLUMN NMAE> =’VALUE’;
SQL> DELETE FROM EMPVIEW WHERE EMPLOYEE_NAME='SRI';
1 ROW DELETED.

SQL> SELECT * FROM EMPVIEW;
EMPLOYEE_N EMPLOYEE_NO DEPT_NAME DEPT_NO
---------- ---------- ---------- ----------
RAVI 124 ECE 89
VIJAY 345 CSE 21
RAJ 98 IT 22
GIRI 100 CSE 67

UPDATE STATEMENT:
SYNTAX:
AQL>UPDATE <VIEW_NAME> SET< COLUMN NAME> = <COLUMN NAME>
+<VIEW> WHERE <COLUMNNAME>=VALUE;
SQL> UPDATE EMPKAVIVIEW SET EMPLOYEE_NAME='KAVI' WHERE
EMPLOYEE_NAME='RAVI';
1 ROW UPDATED.

SQL> SELECT * FROM EMPKAVIVIEW;
EMPLOYEE_N EMPLOYEE_NO DEPT_NAME DEPT_NO
---------- ---------- ---------- ----------
KAVI 124 ECE 89
VIJAY 345 CSE 21
RAJ 98 IT 22
GIRI 100 CSE 67

DROP A VIEW:
SYNTAX:
SQL> DROP VIEW <VIEW_NAME>
EXAMPLE
SQL>DROP VIEW EMPVIEW;
VIEW DROPPED

EX.NO.8	PROCEDURES

AIM
To execute procedure in PL/SQL.

PROCEDURE:
Stored Procedures are created to perform one or more DML operations on Database. It is group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not return a value.

The parameters, which can be the following three types:
- IN: It is the default parameter that will receive input value from the program
- OUT: It will send output value to the program
- IN OUT: It is the combination of both IN and OUT. Thus, it receives from, as well as sends a value to the program

SYNTAX:

```
CREATE or REPLACE PROCEDURE name (parameters)
AS
variables;
BEGIN;
//statements;
END;
```

1. PROGRAM:

Creating a Stored Procedure That Uses Parameters

```
-- including OR REPLACE is more convenient when updating a subprogram
-- IN is the default for parameter declarations
```

CREATE OR REPLACE PROCEDURE award_bonus (emp_id IN NUMBER, bonus_rate IN NUMBER)
AS

```
-- declare variables to hold values from table columns, use %TYPE attribute
  emp_comm       employees.commission_pct%TYPE;
  emp_sal        employees.salary%TYPE;

-- declare an exception to catch when the salary is NULL
  salary_missing  EXCEPTION;
BEGIN  -- executable part starts here
```

```
-- select the column values into the local variables
  SELECT salary, commission_pct INTO emp_sal, emp_comm FROM employees
   WHERE employee_id = emp_id;

-- check whether the salary for the employee is null, if so, raise an exception
  IF emp_sal IS NULL THEN
    RAISE salary_missing;
  ELSE
    IF emp_comm IS NULL THEN

-- if this is not a commissioned employee, increase the salary by the bonus rate
-- for this example, do not make the actual update to the salary
   UPDATE employees SET salary = salary + salary * bonus_rate
   WHERE employee_id = emp_id;
      DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id || ' receives a bonus: '
                  || TO_CHAR(emp_sal * bonus_rate) );
    ELSE
      DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id
                  || ' receives a commission. No bonus allowed.');
    END IF;
  END IF;
EXCEPTION  -- exception-handling part starts here
  WHEN salary_missing THEN
     DBMS_OUTPUT.PUT_LINE('Employee ' || emp_id ||
                 ' does not have a value for salary. No update.');
  WHEN OTHERS THEN
     NULL; -- for other exceptions do nothing
END award_bonus;
/
```

EXECUTION OG THE PROCEDURE:

-- the following BEGIN..END block calls, or executes, the award_bonus procedure

-- using employee IDs 123 and 179 with the bonus rate 0.05 (5%)

```
BEGIN
 award_bonus(123, 0.05);
 award_bonus(179, 0.05);
END;
/
```

OUTPUT:

Employee 123 received a bonus: 325

Employee 179 receives a commission. No bonus allowed.

EX.NO.9	FUNCTIONS

AIM:
To execute function in PL/SQL.

FUNCTION:

- Functions are a standalone block.
- Function use RETURN keyword to return the value, and the datatype of this is defined at the time of creation.
- A Function should either return a value or raise the exception, i.e. return is mandatory in functions.

SYNTAX:

```
CREATE OR REPLACE FUNCTION
<procedure_name>
   (
    <parameter1 IN/OUT <datatype>
     ..
     .
    )
    RETURN <datatype>
[IS | AS ]
    <declaration_part>
BEGIN
     <execution part>
EXCEPTION
     <exception handling part>
END;
```

PROGRAM:

```
CREATE OR REPLACE FUNCTION last_first_name (empid NUMBER)
  RETURN VARCHAR2 IS
  lastname   employees.last_name%TYPE; -- declare a variable same as last_name
  firstname  employees.first_name%TYPE; -- declare a variable same as first_name
BEGIN
  SELECT last_name, first_name INTO lastname, firstname FROM employees
    WHERE employee_id = empid;
  RETURN ( 'Employee: ' || empid || ' - ' || UPPER(lastname)
                        || ', ' || UPPER(firstname) );
END last_first_name;
/
```

EXECUTION OF THE FUNCTION:

```
-- you can use the following block to call the function
DECLARE

-- pick an employee ID to test the function
  empid NUMBER := 163;

BEGIN
-- display the output of the function
  DBMS_OUTPUT.PUT_LINE( last_first_name(empid) );
END;
```

OUTPUT:

Statement processed
Employee: 123, KUMAR DEEPAK

II.Write a Function to search an address from the given database.

PROGRAM

```
SQL> create table phonebook (phone_no number (6) primary key,username
varchar2(30),doorno varchar2(10),
street varchar2(30),place varchar2(30),pincode char(6));
Table created.
SQL> insert into phonebook values(20312,'vijay','120/5D','bharathi street','NGO
colony','629002');
1 row created.
SQL> insert into phonebook values(29467,'vasanth','39D4','RK bhavan','sarakkal
vilai','629002');
1 row created.
SQL> select * from phonebook;
```

PHONE_NO USERNAME DOORNO STREET PLACE PINCODE

```
------------------------------ ------------ --------------- --------------------
20312 vijay 120/5D bharathi street NGO colony 629002
29467 vasanth 39D4 RK bhavan sarakkal vilai 629002

SQL> create or replace function findAddress(phone in number) return varchar2 as
address varchar2(100);
begin
select username||','||doorno ||','||street ||','||place||','||pincode into address from
phonebook
where phone_no=phone;
return address;
exception
when no_data_found then return 'address not found';
end;
/
```

Function created.

```
SQL>declare
2 address varchar2(100);
3 begin
4 address:=findaddress(20312);
5 dbms_output.put_line(address);
6 end;
7 /
```

OUTPUT 1:
Vijay,120/5D,bharathi street,NGO colony,629002

PL/SQL procedure successfully completed.

```
SQL> declare
2 address varchar2(100);
3 begin
4 address:=findaddress(23556);
5 dbms_output.put_line(address);
6 end;
7 /
```

OUTPUT2:
Address not found

EX.NO.10 **EMBEDDED SQL**

Embedded SQL is a method of combining the computing power of a programming language and the database manipulation capabilities of SQL. Embedded SQL statements are SQL statements written inline with the program source code of the host language. The embedded SQL statements are parsed by an embedded SQL pre processor and replaced by host-language calls to a code library. The output from the pre processor is then compiled by the host compiler. This allows programmers to embed SQL statements in programs written in any number of languages such as C/C++, COBOL and Fortran.

The SQL standards committee defined the embedded SQL standard in two steps: a formalism called Module Language was defined, then the embedded SQL standard was derived from Module Language. The SQL standard defines embedding of SQL as embedded SQL and the language in which SQL queries are embedded is referred to as the host language. A popular host language is C. The mixed C and embedded SQL is called Pro*C in Oracle and Sybase database management systems. In the Postgres SQL database management system this pre compiler is called ECPG. Other embedded SQL pre compilers are Pro*Ada, Pro*COBOL, Pro*FORTRAN, Pro*Pascal, and Pro*PL/I.

SQL can be embedded within procedural programming languages. These language (sometimes referred to as 3GLs) include C/C++, Cobol, Fortran, and Ada . Thus the embedded SQL provides the 3GL with a way to manipulate a database, supporting:

- highly customized applications
- background applications running without user intervention
- database manipulation which exceeds the abilities of simple SQL
- applications linking to Oracle packages, e.g. forms and reports
- applications which need customized window interface

❖ Approach: Embed SQL in the host language.

- A preprocessor converts the SQL statements into special API calls.
- Then a regular compiler is used to compile the code.

❖ Language constructs:

- Connecting to a database:
  EXEC SQL CONNECT
- Declaring variables:
  EXEC SQL BEGIN (END) DECLARE SECTION
- Statements:
  EXEC SQL Statement

In the host program:

```
EXEC SQL BEGIN DECLARE SECTION

char c_ sname[20];
```

long c_ sid;

short c_ rating;

float c_ age;

EXEC SQL END DECLARE SECTION

- Two special “error” variables:
  - SQLCODE (long, is negative if an error has occurred)
  - SQLSTATE (char[6], predefined codes for common errors)

EX.NO.11 PAYROLL PROCESSING SYSTEM

ABOUT PROJECT

A Payroll processing application maintains the payroll data. The payroll data includes employee working hours, employee name, employee number, salary details and deductions. This system computes Net Pay for an employee.

MODULES:

1. **LOGIN** : Provides security and to control the user level of access.
2. **EMPLOYEE INFORMATION:** Maintain the employee information like employee number, employee name, address, age, gender
3. **PAYROLL DATA :** Maintain employee salary details, working hours, deductions.
4. **NETPAY CALCULATION:** Compute the Net Salary based on the payroll data.

SOFTWARE REQUIREMENTS:

This application can be developed using Microsoft visual studio as a front end and SQL server as backend.

EX.NO.12	BANKING SYSTEM

ABOUT THE PROJECT:

Banks have traditionally been in the forefront of harnessing technology to improve their products, services and efficiency. They have, over a long time, been using electronic and telecommunication networks for delivering a wide range of value added products and services. This application is developed to maintain the Customer details, Account details and also the Transaction details.

MODULES:

1. **LOGIN** : Provides security and to control the user level of access.
2. **CUSTOMER DETAILS:** Maintain the details of customer name, customer address.
3. **ACCOUNT DETAILS:** Maintain the account details of the customer.
4. **TRANSACTION DETAILS:** The transaction details include both withdraw and deposit.

SOFTWARE REQUIREMENTS

This application can be developed using Microsoft visual studio as a front end and SQL server as backend.

EX.NO.13	LIBRARY MANAGEMENT SYSTEM

ABOUT THE PROJECT

Library management system aims in developing a computerized system to maintain all the daily work of library. This project has many features which are generally not available in normal library management systems like facility of user login for staff and students. This facility makes the students to see the list of books borrowed from the library in their login. The librarian can generate various reports such as student report, issue and return report and book report from their admin account. This project is developed to help the students as well as librarian to maintain the library in the best way and also to reduce the human efforts.

MODULES

1. **LOGIN**: Provides security and to control the user level of access.
2. **ADD NEW BOOK:** Add new book to the library database.
3. **ADD NEW MEMBER:** Add new user to create account.
4. **ISSUE BOOK:** Maintain the details of the user to whom the book is issued.
5. **RETURN BOOK:** Maintain the details of the user and the book returning date.
6. **FINE CALCULATION:** If the book returned after the due date then fine will be calculated for the user.

SOFTWARE REQUIREMENTS

This application can be developed using Microsoft visual studio as a front end and SQL server as backend.

EX.NO.14 **STUDENT INFORMATION SYSTEM**

ABOUT THE PROJECT

Student information system means the information system for maintaining and providing student information. This is maintained in all the schools, colleges, universities and any other education institutions. Some of them are paper based; heavy manual work is involved in managing and maintaining information such as student personal records. However, recently, most schools maintain the record in database. In this system, the user can add, edit, search, delete and view student's information. All the records are listed below in the system which makes the user, easy to view the information of each and every student. This system is easy to operate and understand by the user.

MODULES

1. **LOGIN** : Provides security and to control the user level of access.
2. **STUDENT ADDITION/UPDATION/DELETION:** Add/update/delete the record in the student database.
3. **FEES DETAILS :** Updation of fees details for each student in the database
4. **MARK ENTRY:** Updation of marks for each student in the database
5. **VIEW STUDENT DETAILS:** View the details of all the student records.

SOFTWARE REQUIREMENTS

This application can be developed using Microsoft visual studio as a front end and SQL server as backend.