SDM COLLEGE OF ENGINEERING AND TECHNOLOGY

Dhavalagiri, Dharwad-580002, Karnataka State, India.

Email: cse.sdmcet@gmail.com

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A Report

On

Lab Work

COURSE CODE: 22UCSC501   COURSE TITLE: DBMS
SEMESTER: V   DIVISION: B
COURSE TEACHER: DR. U. P. KULKARNI

[Academic Year- 2025-26]

Date of Submission: 12-12-2025

Submitted
By

Mr. SIDRAMAPPA HALANNAVAR   USN: 2SD23CS099

Table of content:

TOPICS

Problem Statement: Employee–Project Management System

The Employee–Project Management System is designed to store and manage information about employees and the projects they are working on. The system captures basic employee details, project details, and the relationship between them using an assignment table. This helps track employees working on projects and supports efficient querying for project management.S

Entities:

- Employee (empno# , name, email, phone, dob)
- Project (projectno# , projectname, chiefarchitect)

Relationship:

- Assigned_To (empno# , projectno#)
- Many-to-many between Employee and Project.

Relational Schema

Employee(empno#, Name, sex, phone, DOB)
Project(projectno#, projectname , chiefarchitect)
Assigned_To(empno# , projectno#)

Query-1: Create a table Employee.

SQL statement:

```
 SQL> create table employee(

              empno integer not null,

              constraint EMPLOYEE_PK_VIOLATION

              primary key(empno),

              name char(20) not null,

              sex char(1) not null

              constraint EMPLOYEE_SEX_VIOLATION
```

```
        check(sex in ('m','f')),
        phone integer null,
        DOB date not null
    );
```

Output: Table created

Learning Outcomes:

- Implementing primary key constraints and naming conventions
- Applying check constraints for data validation

Query-2: Create a table Project.

SQL statement:

```
SQL> create table project(
        projectno integer not null,
        projectname char(20) not null,
        chiefarchitect char(20) default 'smk' not null,
        constraint PROJECT_PK_VIOLATION
        primary key(projectno)
   );
```

Output: Table created

Learning Outcomes:

- Using default values in table definitions

Query-3: Create a table Assigned_To.

SQL statement:

```
SQL> create table assigned_to(
             empno integer not null,
             projectno integer not null,
             constraint ASSIGNED_TO_PK_VIOLATION
             primary key(empno,projectno),
            constraint ASSIGNED_TO_FK_EMP_VIOLAION
            foreign key(empno)
            references Employee(empno),
            constraint ASSIGNED_TO_FK_PROJ_VIOLAION
            foreign key(projectno)
            references Project(projectno)
     );
```

Output: Table created

Learning Outcomes:

- Managing null and not null constraints for data integrity
- Using foreign keys and constraints to ensure data integrity and referential integrity

Query-4: Describe all tables.

SQL statement:

```
SQL> desc Emloyee
```

Output:

```
Name                                      Null?          Type
----------------------------------------- -------------- -----------
EMPNO                                     NOT NULL       NUMBER(38)
 NAME                                     NOT NULL       CHAR(20)
 SEX                                      NOT NULL       CHAR(1)
 PHONE                                    NOT NULL       NUMBER(38)
 DOB                                      NOT NULL       DATE
```

SQL statement:

SQL> desc project

Output:

```
 Name                                     Null?          Type
 ---------------------------------------- -------------- ----------
 PROJECTNO                                NOT NULL       NUMBER(38)
 PROJECTNAME                              NOT NULL       CHAR(20)
 CHIEFARCHITECT                           NOT NULL       CHAR(20)
```

SQL statement:

SQL> desc Assigned_To

Output:

```
 Name                             Null?           Type
 -------------------------------- --------------- -------------
 EMPNO                            NOT NULL        NUMBER(38)
 PROJECTNO                        NOT NULL        NUMBER(38)
```

Learning Outcomes:

- Understanding column attributes including data types, null constraints, and sizes

Query-5: Insert values in Tables.

SQL statement:

```
SQL>insert all
 into Employee values(1,'Sidram','m',1276819,'05-aug-05');
 into Employee values(2,'Hemanth','m',1276820,'17-feb-05');
 into Employee values(3,'Kiran','m',1276822,'16-apr-05');
select * from dual;
```

Output:3 rows created

SQL statement:

```
SQL>insert into Project values(1,'DBMS','upk');
insert into Project(projectno,projectname) values(2,'CN');
```

Output:2 rows created

SQL statement:

```
 SQL> insert into Assigned_to values(&Empno,&Projectno);
```

Output:

```
Enter value for empno: 2
Enter value for projectno: 1
1 rows created
```

SQL statement:

```
SQL> insert into Assigned_to values(1,2);
```

Output:

```
1 row created.
```

Learning Outcomes:

- Using insert all syntax for multiple row insertion
- Using substitution variables for dynamic data entry

Query-6: Display name, employeeno of all employees.

SQL statement:

```
SQL>select empno, name
from employee;
```

Output:

```
EMPNO   NAME
-----------  --------
   1        Sidram
   2        Hemanth
   3        Kiran
```

Learning Outcomes:

- Understanding result set formatting and column ordering

Query-7: Display details of all male employees born on or after a certain date.

SQL Statement:

```
SQL> select *
   from employee
   where sex='m'
```

```
    and dob>='01-mar-05';
```

Output:

```
EMPNO  NAME             S    PHONE    DOB
----------  ------------------ --  ------------  ---------------
         1     Sidram              m    1276819  05-AUG-05
         3     Kiran              m    1276822  16-APR-05
```

Learning Outcomes:

- Using where clause with multiple conditions
- Applying comparison operators for date and character data filtering

Query-8: Write a SQL statement to obtain the EmpNo of all Employees working on project 1

SQL Statement:

```
SQL> select empno
    from assigned_to
    where projectno = 1;
```

Output:

```
     EMPNO
    ----------
         3
```

Learning Outcomes:

- Using where clause for result filtering

Query-9: Write a SQL statement to get the details of employees working on project 1.

SQL Statement:

```
SQL> select e.*
   from employee e, assigned_to at
   where e.empno=at.empno
   and at.projectno=1;
```

Output:

```
EMPNO  NAME              S    PHONE   DOB
---------- -------------------- -- ------------- ---------------
         3    Kiran              m    1276822   16-APR-05
```

Learning Outcomes:

- Using cartesian product and join conditions to avoid spurious tuples

Query-10: Write a SQL statement to get details of Employees working on 'DBMS' project.

SQL statement:

```
SQL> select e.*
   from employee e, project p, assigned_to at
   where e.empno=at.empno
   and p.projectno=at.projectno
   and p.projectname='DBMS';
```

Output:

```
 EMPNO  NAME             S   PHONE   DOB
----------- -------------------- --  ----------   ------------------------
       1     Sidram             m  1276819  05-AUG-05
       3     Kiran            m  1276822  16-APR-05
```

Learning Outcomes:

- Using cartesian product and join conditions to filter only the required results

Query-11: Modify schema to accommodate penalty paid by employees for various violations.

Relational schema:

Employee(Empno#, Name, Sex, Phone, DOB)

Penalty(Empno#, DateTime#, penaltyAmount, Reason)

SQL statement:

```
SQL> create table penalty(
   empno integer not null,
   Datetime timestamp,
   penaltyAmount integer not null,
   Reason char(50) not null,
   constraint PENALTY_PK_VIOLATION
   primary key(empno, Datetime),
   constraint PENALTY_FK_EMP_VIOLAION
   foreign key(empno)
  references Employee(empno)
  );
```

Output:

Table created.

Learning Outcomes:

- Using timestamp datatype for datetime storage
- Using composite primary key and foreign key for referential integrity

Query-12: List details of all employees who have paid Penalty so far.

SQL statement:

```
SQL> select e.*, p.penaltyAmount FINE
  from employee e, penalty p
  where e.empno = p.empno;
```

Output:

```
EMPNO FULL_NAME    S    PHONE      DOB           FINE
---------- -------------------- -- -------------- --------------- -----------
       2    Hemanth             m    1276820    17-FEB-05     5000
       3    Kiran             m    1276822    16-APR-05    1000
```

Learning Outcomes:

- Using inner join to filter only the employees who have paid penalty.

Query-13: Find for each employee the penalty incurred.

SQL Statement:

```
SQL> select e.*, p.penaltyAmount FINE
   from employee e, penalty p
   where e.empno=p.empno(+)
   order by e.empno;
```

Output:

```
EMPNO  FULL_NAME   S   PHONE    DOB          FINE
---------- ------------------- -- ------------- --------------- ----------------
      1     Sidram          m   1276819   05-AUG-05
      2     Hemanth          m   1276820   17-FEB-05     5000
      3     Kiran          m   1276822   16-APR-05     1000
```

Learning Outcomes:

- Using left outer join condition to include all employees' penalty, create a null row if no match found and match with that employee.
- Use order by to format the result according to empno

Query-14: Rewrite Query-10 using 'IN' operator.

SQL Statement:

```
SQL> select *
   from employee
   where empno in (
   select empno
   from assigned_to
   where projectno in (
   select projectno
   from project
   where projectname='DBMS')
  );
```

Output:

```
EMPNO  FULL_NAME    S    PHONE    DOB
----------   -------------------  ---  ------------  ----------------
        1          Sidram            m    1276819   05-AUG-05
        3          Kiran          m    1276822   16-APR-05
```

Learning Outcomes:

- Understanding the cons of cartesian product and using IN operator for efficient querying.
- Understanding how subquery execution works.

Query-15: Write a SQL statement to get details of employees working on both projects 1 and 2

SQL Statement:

```
SQL> select *
  from employee
   where empno IN(
    select empno
    from Assigned_to
   where projectno=1)
   And empno IN(
   select empno
```

```
    from Assigned_to
  where projectno=2);
```

Output:

```
EMPNO  FULL_NAME    S   PHONE    DOB
----------  -------------------- ---  ------------  -----------------
1           Sidram             m    1276819   05-AUG-05
```

Learning Outcomes:

- Using IN subqueries along with AND condition to find intersection of employees working on different projects.

Query-16: Display the details of all employees who are working on all projects available in the organization.

SQL Statement:

```
SQL>select e.*
From employee e
Where not exists(
        Select p.projectNo
        From project p
        Where projectNo not in(
                Select at.projectNo
                From Assigned_To at
                Where at.empNo=e.empNo
        )
);
```

Output:

```
EMPNO  FULL_NAME    S   PHONE    DOB
----------  -------------------- ---  ------------  -----------------
1           Sidram              m    1276819   05-AUG-05
```

Learning Outcomes:

- Understands implementation of **Relational Division in SQL**.
- Uses **NOT EXISTS with NOT IN** to compare sets.
- Learns to retrieve records that satisfy **“for all” conditions**.

Query-17:

I. Get the EmpNo of employees working on ProjectNo=1

```
 SQL> select empNo
         From ASSIGNED_To
         Where ProjectNo=1;
```

Output:

```
    EMPNO
   ----------
        3
```

Learning Outcomes:

- Learns to use SELECT with WHERE clause for conditional filtering.
- Retrieves attribute values based on foreign key condition.

II. Get the details of employees(both empno and name) working on project 1

```
SQL> select e.empno, e.empname

From employee e, assigned_to at

Where e.empno=at.empno

And at.projectNo=1;
```

Output:

```
EMPNO  EMPNAME

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

         3    Kiran
```

Learning Outcomes:

- Understands **joining multiple tables using join condition**.
- Retrieves data from **related tables using foreign keys**.

III. Obtain details of employees working on 'DBMS' project.

```
SQL> select e.*

From employee e, project p, assigned_to at

Where e.empno=at.empno

And at.projectno=p.projectNo

And p.projectName='DBMS';
```

Output:

```
EMPNO  FULL_NAME    S    PHONE    DOB

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

        1          Sidram            m    1276819   05-AUG-05

        3          Kiran          m    1276822   16-APR-05
```

Learning Outcomes:

- Learns multi-table joins using three tables.
- Uses attribute-based filtering (projectName).
- Understands mapping between Employee–Project–Assignment tables.

IV. Gather details of employees working on both project 1 and 2

```
SQL> select *
   from employee
    where empno IN(
     select empno
     from Assigned_to
    where projectno=1)
    And empno IN(
    select empno
     from Assigned_to
   where projectno=2);
```

Output:

```
EMPNO  FULL_NAME    S   PHONE    DOB
----------  -------------------- ---  ------------  -----------------
1          Sidram               m    1276819   05-AUG-05
```

Learning Outcomes:

- Uses **set intersection logic using IN clause**.
- Understands how to find **common records across multiple conditions**.

V. Find empno of employees who work on all projects that employee 3 works on.

```
SQL> select empno
From assigned_to a
Where not exists(
        (Select projectNo from assigned_to
        Where empNo=3)
        Minus
        (select projectNo from assigned_to
        Where empNo=a.empNo
        )
)
Minus
(select empNo from employee
Where empNo=3);
```

Output:

```
     EMPNO
----------
         1
```

Learning Outcomes:

- Implements **Relational Division using NOT EXISTS and MINUS**.
- Compares two project sets for **subset matching**.
- Understands **advanced set-based queries**.

VI. Find empNo of employees who do not work on project 1.

```
SQL> select empNo from assigned_to
```

```
Minus

Select empNo from assigned_to where projectNo=1;
```

Output:

```
    EMPNO
    ----------
       3
```

Learning Outcomes:

- Uses **MINUS operator for set difference**.
- Learns how to **exclude records using set operations**.

VII. Get empNo of employees who work on all projects.

```
SQL> select empNo from assigned_to a
Where not exists(
        (select projectNo from project)
        Minus
        (Select projectNo from assigned_to
        Where empNo=a.empNo)
);
```

Output:

```
        EMPNO
----------
             1
```

Learning Outcomes:

- Applies **Relational Division using full project set**.
- Uses **NOT EXISTS with MINUS**.

- Identifies employees with **complete project coverage**.

VIII. List empNo of employees other than employee 1 who work on atleast one of the projects that employee 1 works on.

```
SQL>select empNo

From assigned_to

Where projectNo in(

        Select projectNo

        From assigned_to

        Where empNo=1

)

Minus

Select empNo froom assigned_to where empNo=1;
```

Output:

```
     EMPNO
----------
         2
         3
```

Learning Outcomes:

- Uses **nested subqueries with IN**.
- Learns **exclusion using MINUS**.
- Understands **partial set matching (at least one condition)**.

Query-18:

i. **Retrieve the names of instructors along with the names of courses they teach in each semester.**

SQL Statement:

```
select i.Inst_Name, c.Course_Name, t.Semester
```

```
from Instructor i

join Teaches t on t.Inst_ID=i.Inst_ID

join Course c on c.Course_ID=t.Course_ID

order by i.Inst_Name, t.Semester;
```

Output:

```
INST_NAME            COURSE_NAME            SEMESTER
-------------------- -------------------- ----------
C.C.D                CN                            3
U.P.K                DIP                           4
U.P.K                DBMS                          5
```

Learning Outcomes:

- Understand how to perform multi-table joins across Instructor, Teaches, and Course tables.
- Learn to combine teaching assignments with related course and instructor details.
- Apply ORDER BY to organize multi-table results meaningfully.
- Gain practical experience in retrieving relationship-based academic data (Instructor–Course–Semester).

ii. List names of students and instructors in the same department who are associated with the same course, including the course name.

SQL Statement:

```
select i.Inst_Name,s.Stud_Name,i.Dept,c.Course_Name

from Student s

join Enrolled e on e.Stud_ID=s.Stud_ID

join Course c on c.Course_ID=e.Course_ID

join Teaches t on t.Course_ID=c.Course_ID
```

join Instructor i on i.Inst_ID=t.Inst_ID;

Output:

```
INST_NAME        STUD_NAME        DEPT        COURSE_NAME
---------------- --------------- ----------- --------------
U.P.K            Hemanth           CSE          DBMS
U.P.K            Tejas            CSE         DBMS
U.P.K            Hemanth           CSE          DIP
U.P.K            Tejas            CSE         DIP
C.C.D            Prajwal          ECE         CN
```

Learning Outcomes:

- Learn to connect four related tables (Student, Enrolled, Course, Teaches, Instructor) using JOINs.
- Understand how to retrieve student–instructor–course associations.
- Identify relationships across departments and verify department-level matching.
- Gain experience in solving complex queries involving multiple joins and shared attributes.

Query-19:

a. Study create table syntax using select statement in terms of constraints of original table.

SQL>create table e1

As select * from employee;

Output:

Table created.

Learning Outcome:

- Learns **table creation using existing table structure**.
- Understands that **constraints are not copied** during CTAS (Create Table As Select).

b. Study of alter table provision in SQL.

```
SQL>alter table employee

Add(hometown char(40) null);
```

Output:

```
Table altered
```

Learning Outcomes:

- Understands **modification of table structure after creation**.
- Learns to **add shivattributes dynamically**.

c. Create synonym for table.

```
SQL> create synonym e2 for employee;
```

Output:

```
Synonym created.
```

Learning Outcomes:

- Learns how to create **alternate names (aliases)** for database objects.
- Understands **object abstraction and simplification**.

d. Study of the following:

  i. Update

```
SQL>update employee

Set phone = 8298020201

Where empno=3;
```

Output: 1 row updated.

Learning Outcomes:

- Learns to **modify existing records**.

- Understands **row-level updates using WHERE clause**.

ii. Order by ascending and descending

SQL>select * from employee

Order by empname;

Output:

```
     EMPNO NAME                 S       PHONE DOB
---------- -------------------- - ---------- ---------
         2 Hemanth              m    1276820 17-FEB-05
         3 Kiran                m   8298020201 16-APR-05
         1 Sidram               m    1276819 05-AUG-05
```

SQL>select * from employee

Order by dob desc;

Output:

```
     EMPNO NAME                 S       PHONE DOB
---------- -------------------- - ---------- ---------
         1 Sidram               m     1276819 05-AUG-05
         3 Kiran                m   8298020201 16-APR-05
         2 Hemanth              m     1276820 17-FEB-05
```

Learning Outcomes:

- Learns **sorting of query results**.
- Understands **ascending and descending ordering**.

iii. Group by

SQL>select count(projectNo)

From assigned_to

Group by empNo;

Output:

```
     EMPNO COUNT(PROJECTNO)
---------- ----------------
         1                2
         2                1
         3                1
```

Learning Outcomes:

- Understands **aggregation with grouping**.
- Learns **per-group calculations using aggregate functions**.

iv. Deleting Rows from table

SQL>delete from employee

Where empNo=4;

Output: 1 row deleted.

Learning Outcomes:

- Learns **row deletion with conditions**.
- Understands **permanent data removal**.

v. Dropping table

SQL>drop table penalty;

Output: Table dropped.

Learning Outcomes:

- Understands **permanent removal of table structure and data**.

vi. Copying the table

SQL>create table e3 as

```
Select empname,phone
From employee;
```

Output: Table created.

Learning Outcomes:

- Learns to **create tables using selected attributes only**.
- Understands **projection operation in SQL**.

vii. Renaming the tables

```
SQL>alter table employee
Rename to employees;
```

Output: Table altered.

Learning Outcomes:

- Understands how to **rename database objects safely**.

viii. Between

```
SQL>select * from employee
Where dob between '01-APR-05' and '01-AUG-05';
```

Output:

```
     EMPNO NAME                 S      PHONE DOB
---------- -------------------- - ---------- ---------
         3 Kiran                m 8298020201 16-APR-05
```

Learning Outcomes:

- Learns **range-based filtering using BETWEEN**.
- Applies **date-based conditional queries**.

ix. Like

SQL>select empname

From employee

Where empname like '%in%';

Output:

```
NAME
--------------------
Hemanth
```

Learning Outcomes:

- Understands **pattern matching using wildcards**.
- Learns **partial string searching in SQL**.

x. Study of views with check option, without check option

I. Create view with check option.

SQL>create view emp_info as

Select empno,empname,sex,dob

From employee

Where dob > '01-feb-05'

With check option;

Output:

View created.

SQL>Insert into emp_info values(10,'New','m','01-jan-05');

Output:

```
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause
violation
```

Learning Outcomes:

- Understands **data validation through CHECK OPTION**.
- Ensures that **inserted data satisfies view condition**.

II. Create views without check constraint

SQL>create view emp2_info as

Select empNo, empName, sex, dob

From employee

Where dob > ’01-feb-05’;

Output:

View created.

SQL>Insert into emp2_info values(10,’shiv’,’m’,’01-jan-05’);

Output:

1 row created.

Learning Outcomes:

- Learns how **views allow flexible data insertion without restriction**.
- Understands **difference between restricted and unrestricted views**.

e. Study of null option, changing the column heading in SQL result

SQL>select * from employee
Where phone is null;

Output:

```
     EMPNO NAME                 S      PHONE DOB
---------- -------------------- - ---------- ---------
        10 shiv                  m            01-JAN-05
```

SQL>select empno EMPID, empname
From employee;

Output:

```
     EMPID NAME
---------- --------------------
        10 shiv
         1 Sidram
         2 Hemanth
         3 Kiran
```

Learning Outcomes:

- Learns to **handle NULL values in SQL**.
- Understands **renaming column headings using aliases**.

f. Study of having clause.

SQL>select empNo, count(projectNo)

From assigned_to

Group by empNo

Having count(projectNo) >= 2;

Output:

```
     EMPNO COUNT(PROJECTNO)
---------- ----------------
         1                2
```

Learning Outcomes:

- Learns **group-level filtering using HAVING**.
- Understands **difference between WHERE and HAVING**.

Query-20: Study of date and all its functions.

i. Select all employees who were born on ‘THURSDAY’.

SQL Statement:

SQL>select * from employee

Where to_char(dob,’DY’)=’THU’;

Output:

```
     EMPNO NAME                 S      PHONE DOB
---------- -------------------- - ---------- ---------
         2 Hemanth               m    1276820 17-FEB-05
```

Learning Outcomes:

- Understand how to extract day information from a DATE datatype using TO_CHAR.
- Apply conditional filtering on formatted date values.
- Use date format models like 'DY' and 'DAY' effectively.

ii. Show system date as ‘Today is ‘DD-MM-YY’ format’.

SQL Statement:

SQL>select to_char(SYSDATE, ‘“Today is DD-MM-YY”’) TODAY

From dual;

Output:

```
TODAY
-----------------
Today is 06-12-25
```

Learning Outcomes:

- Learn to format system date using TO_CHAR.
- Display custom text along with date inside quotes using escaped characters.
- Use dual table for single-row queries

iii. Display all employees’ birthday in the ‘MONTH-DD’ format.

SQL Statement:

SQL>select empname,to_char(dob,’MONTH-DD’) BIRTH_DATE

From employee;

Output:

```
NAME                    BIRTH_DATE
--------------------    ---------------------------------
shiv                     JANUARY  -01
Sidram                   AUGUST   -05
Hemanth                  FEBRUARY -17
Kiran                   APRIL    -16
```

Learning Outcomes:

- Understand date-to-character conversion with full month names.
- Display dates in user-defined formats using TO_CHAR.
- Learn formatting with padded month names.

iv. Retrieve all employees who have birthday today.

SQL Statement:

SQL>select * from employee

Where to_char(dob,’MM-DD’)=to_char(SYSDATE,’MM-DD’);

Output:

```
     EMPNO NAME                 S      PHONE DOB
---------- -------------------- - ---------- ---------
        10 shiv                  m             14-NOV-03
```

Learning Outcomes:

- Compare date values by converting them into matching string formats.
- Apply the MM-DD format for birthday comparison.
- Understand how SYSDATE assists in dynamic date filtering.

v. Retrieve employees born before a certain month in any year.

SQL Statement:

SQL>select * from employee

Where dob between

To_date('10-MAR-2005','DD-MON-YYYY')

And to_date('25-AUG-2005','DD-MON-YYYY');

Output:

```
     EMPNO NAME                 S      PHONE DOB
---------- -------------------- - ---------- ---------
         1 Sidram               m    1276819 05-AUG-05
         3 Kiran                m 8298020201 16-APR-05
```

Learning Outcomes:

- Use TO_DATE to convert string literals to valid date objects.
- Apply the BETWEEN operator for date range queries.
- Understand date comparisons independent of year.

vi. Retrieve all employees who are above 20 years of age.

SQL Statement:

SQL>select * from employee

Where floor(MONTHS_BETWEEN(SYSDATE,dob)/12)>20;

Output:

```
     EMPNO NAME                 S      PHONE DOB
---------- -------------------- - ---------- ---------
        10 shiv                  m             06-DEC-03
```

Learning Outcomes:

- Calculate age using MONTHS_BETWEEN.
- Use FLOOR() to convert fractional months into full years.
- Apply date difference functions for age-based filtering.

Query-21: Study of PL/SQL.

i. **Create a PL/SQL block to display the total no. of projects as Project Count. Also display ‘too few projects’ if no. of projects < 3 else display ‘sufficient projects’.**

SQL Statement:

```
SQL>declare
    projectCount integer;
    status char(20);
begin
    select COUNT(projectno) into projectCount from project;
    if projectCount<3 then
        status:=’Very few projects.’
    Else
        Status:’Sufficient projects.’
    End if;
    DBMS_output.put_line('No. of projects: '||projectCount);

    DBMS_output.put_line('Status: '||status);

End;

/
```

Output:

```
No. of projects: 2
Status: Very few projects.

PL/SQL procedure successfully completed.
```

Learning Outcomes:

- Write anonymous PL/SQL blocks with variables.
- Use COUNT() with SELECT INTO in PL/SQL.
- Apply IF–ELSE conditions inside PL/SQL.
- Display output using DBMS_OUTPUT.PUT_LINE.

ii. Create a PL/SQL block to display empno, empname of employees who are above 18 years of age.

SQL Statement:

```
SQL> declare

    age integer;

Begin

    DBMS_output.put_line('Employees above 18:')

    For emp_rec in (

            Select empno,empname,dob

            From employee

    )loop
    age:=floor(MONTHS_BETWEEN(SYSDATE,emp_rec.dob)/12);

    If age > 18 then

            DBMS_output.put_line('Employee no: '|| emp_rec.empno ||
' Employee name: '|| emp_rec.empname);

    End if;

    End loop;

End;

/
```

Output:

```
Employees above 18:
Employee no: 10 Employee name: shiv
Employee no: 1 Employee name: Sidram
Employee no: 2 Employee name: Hemanth
Employee no: 3 Employee name: Kiran

PL/SQL procedure successfully completed.
```

Learning Outcomes:

- Use cursor FOR loops to iterate over query results.
- Calculate age inside PL/SQL using date functions.
- Display selective output based on conditions.
- Understand PL/SQL looping and variable handling.

Query-22: Study of Triggers.

i. Prepare a list of all employees containing their ID and old salary whenever there is a increase in salary by more than 10%.

SQL Statement:

```
SQL>create table salaryUpdated(
    Empno integer not null,

    oldSalary integer

)
```

Output: Table created.

SQL Statement:

```
SQL>create trigger t1_salaryUpdated
Before update on employee
For each row
When (new.salary/old.salary > 1.1)
Begin
    Insert into salaryUpdated values(:old.Empno,:old.salary);
End;
```

Output: Trigger created.

SQL Statement:

```
SQL> update employee
Set salary=20000
Where empno=10;
```

Output: 1 row updated.

SQL Statement:

```
SQL> select * from salaryUpdated;
```

Output:

```
     EMPNO  OLDSALARY
---------- ----------
        10      10000
```

Learning Outcomes:

- Create a table to store trigger-generated audit records.
- Understand BEFORE UPDATE row-level triggers.
- Use :OLD and :NEW references.
- Apply conditional trigger firing using the WHEN clause.
- Perform automatic logging of changes through triggers.

Query-23: Study of Stored Procedures.

i. **Create a procedure to add projects by taking projectno, projectname and chief architect as parameters.**

SQL Statement:

```
SQL>create procedure addProjects(
    No in integer,
    Name in varchar2,
    manager in varchar2
)
As
Begin
    Insert into project(projectno,projectname,chiefarchitect)
    Values(no,name,manager);
End;
/
```

Output: Procedure created.

SQL Statement:

```
SQL>exec addProjects(4,’DIP’,’shn’);
```

Output: PL/SQL procedure successfully completed.

SQL Statement:

```
SQL> select * from project;
```

Output:

```
PROJECTNO PROJECTNAME          CHIEFARCHITECT
---------- -------------------- --------------------
         1 DBMS                 upk
         2 CN                   ccd
         3 SE                   sdp
         4 DIP                  shn
```

Learning Outcomes:

- Create stored procedures with IN parameters.
- Insert new rows inside PL/SQL procedural blocks.
- Understand how to execute procedures using EXEC.
- Improve modularity and reuse of common database operations.

Query-24: Study of Functions.

i. **Create a function to get salary of a particular employee by providing their empno as parameter.**

SQL Statement:

```
SQL>create function get_salary(
    emp_id in integer
)
Return integer
Is
    V_salary integer;
Begin
    Select salary into V_salary
    From employee
    Where empno=emp_id;
```

Return v_salary;

Exception

When no_data_found then

Return null;

End;

/

Output: Function created.

SQL Statement:

SQL>select empname, get_salary(empno) salary

From employee;

Output:

```
NAME                      SALARY
-------------------- ----------
Shiv                       20000
Sidram                      50000
Hemanth                     30000
Kiran                     60000
```

Learning Outcomes:

- Create user-defined functions that return values.
- Use SELECT INTO inside function bodies.
- Handle exceptions like NO_DATA_FOUND.
- Call functions inside SQL queries for computation.

Query-25: Study of Cursors and Exceptions.

i. **Write a PL/SQL program using an explicit cursor to count and display the number of male employees in the organization.**

SQL Statement:

```
SQL>
Declare
    counterMaleEmp number:=0;
    sex Employee.sex%type;
    cursor curEmployee is
    select sex from employee;
begin
    open curEmployee
    fetch curEmployee into sex;
    while curEmployee%found loop
            if sex=’m’ then
                    counterMaleEmp:=counterMaleEmp+1;
            end if;
            fetch curEmployee into sex;
    end loop;
    close curEmployee;
    DBMS_output.put_line(‘No. of male employees are:
    ‘||to_char(counterMaleEmp));
end;
/
```

Output:

```
No. of male employees are: 4

PL/SQL procedure successfully completed.
```

Learning Outcomes:

- Declare and use explicit cursors in PL/SQL.
- Fetch data row by row using OPEN, FETCH, CLOSE.
- Use cursor attributes like %FOUND.
- Implement counting logic using conditions.

ii. Write a PL/SQL program to demonstrate exception when no data found, duplicates found, other exceptions.

SQL Statement:

```
SQL>

Declare

    V_empno Employee.empno%type := &empno;

    V_name Employee.empname%type;

    V_salary Employee.salary%type;

    No_Emp_Data_Exception Exception;

Begin

    declare

    Select empname,salary

    Into v_name, v_salary

    From employee

    Where empno=v_empno;

    Exception

    When no_data_found then

            Raise no_emp_data_exception;

    End;
```

```
                    DBMS_output.put_line('Employee found: ');

                    DBMS_output.put_line('Name: '||v_name);

                    DBMS_output.put_line('Salary: '||v_salary);

        Exception

            When no_emp_data_exception then

                    DBMS_output.put_line('Error: Employee with empno
'||v_empno ||' does not exist.');

        End;

        /
```

Output:

```
Error: Employee with empno 25 does not exist.

PL/SQL procedure successfully completed.
```

Learning Outcomes:

- Raise and handle user-defined exceptions.
- Use predefined exceptions such as NO_DATA_FOUND.
- Apply exception handling blocks for safer program execution.
- Display meaningful error messages using DBMS_OUTPUT