INTRODUCTION TO ER MODEL

ER model is represents real world situations using concepts, which are commonly used by people. It allows defining a representation of the real world at logical level.ER model has no facilities to describe machine-related aspects.

In ER model the logical structure of data is captured by indicating the grouping of data into entities. The ER model also supports a top-down approach by which details can be given in successive stages.

Entity: An entity is something which is described in the database by storing its data, it may be a concrete entity a conceptual entity.

Entity set: An entity set is a collection of similar entities.

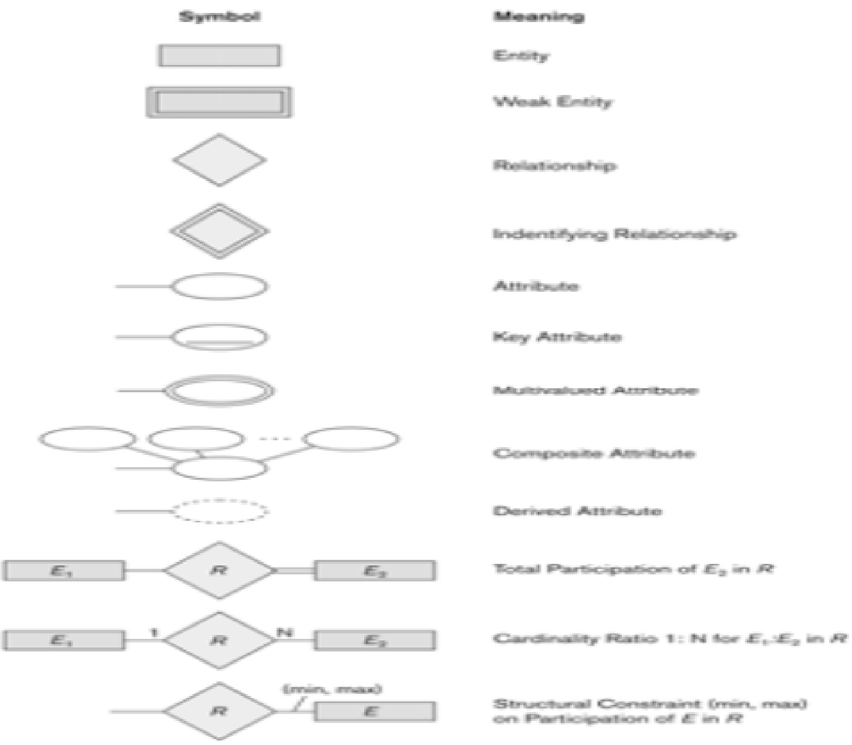
Attribute: An attribute describes a property associated with entities. Attribute will have a name and a value for each entity.

Domain: A domain defines a set of permitted values for a attribute

SYMBOLS IN E-R DIAGRAM

The ER model is represented using different symbols as shown in Fig .a

Figure 3.14
Summary of the notation for ER diagrams.


Symbol
Meaning
Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
Composite Attribute
Derived Attribute
E1
R
E2
Total Participation of E2 in R
E1
1
R
N
E2
Cardinality Ratio 1: N for E1:E2 in R
R
(min, max)
E
Structural Constraint (min, max) on Participation of E in R

EXPERIMENT 1: UNIVERSITY MANAGEMENT SYSTEM ER DIAGRAM

Aim:A university registrar's office maintains data about the following entities: (a) courses, including number, title, credits, syllabus, and prerequisites; (b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom; (c) students, including student-id, name, and program; and (d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled. Construct an E-R diagram forthe registrar's office. Document all assumptions that you make about the mapping constraints.

Output

The main entity sets are student, course, course-offering, and instructor. The entity set course-offering is a weak entity set dependent on course. The assumptions made are :

- A class meets only at one particular place and time. This E-R diagram cannot model a class meeting at different places at different times.
- There is no guarantee that the database does not have two classes meeting at the same place and time

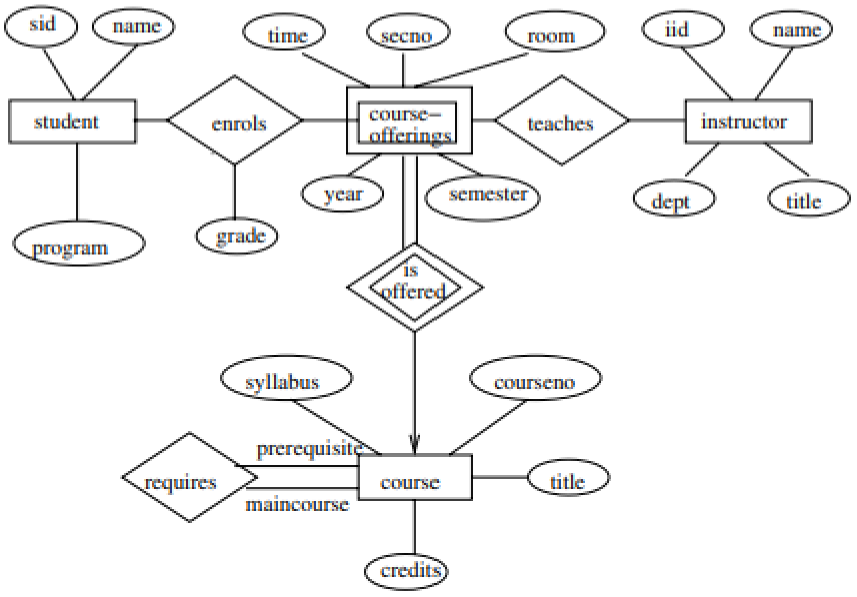


Figure 2.3 E-R diagram for a university.

RESULT

ER diagram has been drawn successfully. By constructing an ER diagram I was able to apply standard design and modelling approach.

INTRODUCTION TO SQL

History of SQL

Dr. E. F. Codd published the paper, "A Relational Model of Data for Large Shared Data Banks", in June 1970 inthe Association of Computer Machinery (ACM) journal, Communications of theACM. Codd's model is nowaccepted as the definitive model for relational database management systems (RDBMS). The language,Structured English Query Language ("SEQUEL") was developed by IBM Corporation, Inc., to use Codd'smodel. SEQUEL later became SQL (still pronounced "sequel"). In 1979, Relational Software, Inc. (now OracleCorporation) introduced the first commercially available implementation of SQL. Today, SQL is accepted as thestandard RDBMS language.

How SQL Works

The strengths of SQL provide benefits for all types of users, including application programmers, databaseadministrators, managers, and end users. Technically speaking, SQL is a data sub-language. The purpose of SQLis to provide an interface to a relational database such as Oracle, and all SQL statements are instructions to thedatabase. In this SQL differs from general-purpose programming languages like C and BASIC. Among thefeatures of SQL are the following:

1. It processes sets of data as groups rather than as individual units.

2. It provides automatic navigation to the data.

3. It uses statements that are complex and powerful individually, and that therefore stand alone.

Flow-control statements were not part of SQL originally, but they are found in the recently accepted optionalpart of SQL, ISO/IEC 9075-5: 1996. Flow-control statements are commonly known as "persistent storedmodules" (PSM), and Oracle's PL/SQL extension to SQL is similar to PSM.

Essentially, SQL lets you work with data at the logical level. You need to be concerned with the implementationdetails only when you want to manipulate the data. For example, to retrieve a set of rows from a table, youdefine a condition used to filter the rows. All rows satisfying the condition are retrieved in a single step and canbe passed as a unit to the user, to another SQL statement, or to an application. You need not deal with the rowsone by one, nor do you have to worry about how they are physically stored or retrieved. All SQL statements usethe optimizer, a part of Oracle that determines the most efficient means of accessing the specified data. Oraclealso provides techniques you can use to make the optimizer perform its job better.

SQL provides statements for a variety of tasks, including:

1. Querying data

2. Inserting, updating, and deleting rows in a table

3. Creating, replacing, altering, and dropping objects

4. Controlling access to the database and its objects

5. Guaranteeing database consistency and integrity

SQL unifies all of the above tasks in one consistent language.

Common Language for All Relational Databases

All major relational database management systems support SQL, so you can transfer all skills you have gainedwith SQL from one database to another. In addition, all programs written in SQL are portable. They can often bemoved from one database to another with very little modification.

Summary of SQL Statements

SQL statements are divided into these categories:

1. Data Definition Language (DDL) Statements

2. Data Manipulation Language (DML) Statements

3. Transaction Control Statements (TCL)

4. Session Control Statement

5. System Control Statement

Managing Tables

A table is a data structure that holds data in a relational database. A table is composed of rows and columns.A table can represent a single entity that you want to track within your system. This type of a table couldrepresent a list of the employees within your organization, or the orders placed for your company's products.

A table can also represent a relationship between two entities. This type of a table could portray the associationbetween employees and their job skills, or the relationship of products to orders. Within the tables, foreign keysare used to represent relationships.

Creating Tables

To create a table, use the SQL command CREATETABLE.

Syntax:

CREATE TABLE <TABLE NAME>(<FIELD NAME ><DATA TYPE><[SIZE]>,..........)

Altering Tables

Alter a table in an Oracle database for any of the following reasons:

1. To add one or more new columns to the table

2. To add one or more integrity constraints to a table

3. To modify an existing column's definition (datatype, length, default value, and NOTNULL

4. integrity constraint)

5. To modify data block space usage parameters (PCTFREE, PCTUSED)

6. To modify transaction entry settings (INITRANS, MAXTRANS)

7. To modify storage parameters (NEXT, PCTINCREASE, etc.)

8. To enable or disable integrity constraints associated with the table

9. To drop integrity constraints associated with the table

When altering the column definitions of a table, you can only increase the length of an existing column, unlessthe table has no records. You can also decrease the length of a column in an empty table. For columns of datatype CHAR, increasing the length of a column might be a time consuming operation that requires substantialadditional storage, especially if the table contains many rows. This is because the CHAR value in each row mustbe blank-padded to satisfy the new column length.

If you change the datatype (for example, from VARCHAR2 to CHAR), then the data in the column does notchange. However, the length of new CHAR columns might change, due to blank-padding requirements.

Altering a table has the following implications:

1. If a new column is added to a table, then the column is initially null. You can add a column with a

NOT NULL constraint to a table only if the table does not contain any rows.

2. If a view or PL/SQL program unit depends on a base table, then the alteration of the base table mightaffect the dependent object, and always invalidates the dependent object.

Privileges Required to Alter a Table

To alter a table, the table must be contained in your schema, or you must have either the ALTERobject privilege for the table or the ALTER ANY TABLE system privilege.

Dropping Tables

Use the SQL command DROP TABLE to drop a table. For example, the following statement drops the

EMP_TAB table:

If the table that you are dropping contains any primary or unique keys referenced by foreign keys to other tables,and if you intend to drop the FOREIGN KEY constraints of the child tables, then include the CASCADE optionin the DROP TABLE command.

Oracle Built-In Datatypes

A datatype associates a fixed set of properties with the values that can be used in a column of a table or in anargument of a procedure or function. These properties cause Oracle to treat values of one datatype differentlyfrom values of another datatype. For example, Oracle can add values of sNUMBER datatype, but not values of

RAW datatype.

Oracle supplies the following built-in data types:character data types

- CHAR

- NCHAR

- VARCHAR2 and VARC

- NVARCHAR2

- CLOB

- NCLOB

- LONG

1. NUMBER datatype

2. DATE datatype

3. Binary datat ypes

- BLOB

- BFILE

- RAW

- LONG RAW

Another datatype, ROWID, is used for values in the ROWIDpseudocolumn, which represents the unique addressof each row in a table.

Table summarizes the information about each Oracle built-in datatype.

Summary of Oracle Built-In Data types

Using Character Data types

Use the character data types to store alphanumeric data.

1. CHAR and NCHAR data types store fixed-length character strings.

2. VARCHAR2 and NVARCHAR2 data types store variable-length character strings. (The VARCHARdataty is synonymous with the VARCHAR2 datatype.)

3. CLOB and NCLOB data types store single-byte and multi byte character strings of up to four gigabytes.

4. The LONG datatype stores variable-length character strings containing up to two gigabytes, but withmany restrictions.

5. This data type is provided for backward compatibility with existing applications; in general, newapplications should use CLOB and NCLOB data types to store large amounts of character data.

When deciding which datatype to use for a column that will store alphanumeric data in a table,consider the following points of distinction:

Space Usage

1. To store data more efficiently, use the VARCHAR2 datatype. The CHAR data type blank-pads and storestrailing blanks up to a fixed column length for all column values, while the VARCHAR2 datatype doesnot blank-pad or store trailing blanks for column values.

2. Use the CHAR data type when you require ANSI compatibility in comparison semantics (when trailingblanks are not important in string comparisons). Use the VARCHAR2 when trailing blanks are importantin string comparisons.

Comparison Semantics

Use the CHAR data type when you require ANSI compatibility in comparison semantics (when trailing blanksare not important in string comparisons). Use the VARCHAR2 when trailing blanks are important in stringcomparisons.

Future Compatibility

1. The CHAR and VARCHAR2 data types are and will always be fully supported. At this time, theVARCHAR datatype automatically corresponds to the VARCHAR2 datatype and is reserved for futureuse.

CHAR, VARCHAR2, and LONG data is automatically converted from the database character set tothe character set defined for the user session by the NLS_LANGUAGE parameter, where these aredifferent.

Using the NUMBER Datatype

Use the NUMBER datatype to store real numbers in a fixed-point or floating-point format. Numbers using thisdata type are guaranteed to be portable among different Oracle platforms, and offer up to 38 decimal digits ofprecision. You can store positive and negative numbers of magnitude 1 x 10^-130 to 9.99...x10^125, as well aszero, in a NUMBER column.

For numeric columns you can specify the column as a floating-point number:

Column_name NUMBER

Or, you can specify a precision (total number of digits) and scale (number of digits to the right of the decimalpoint):

Column_name NUMBER (<precision>, <scale>)

Although not required, specifying the precision and scale for numeric fields provides extra integrity checking oninput. If a precision is not specified, then the column stores values as given. Table shows examples of how datawould be stored using different scale factors.

Using the DATE Datatype

Use the DATE datatype to store point-in-time values (dates and times) in a table. The DATE datatypestores the century, year, month, day, hours, minutes, and seconds.

Oracle uses its own internal format to store dates. Date data is stored in fixed-length fields of sevenbytes each, corresponding to century, year, month, day, hour, minute, and second.

Date Format

For input and output of dates, the standard Oracle default date format is DD-MON-YY.

For example: '13-NOV-92'

To change this default date format on an instance-wide basis, use the NLS_DATE_FORMAT parameter. Tochange the format during a session, use the ALTER SESSION statement. To enter dates that are not in thecurrent default date format, use the TO_DATE function with a format mask.

For example:

TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')

If the date format DD-MON-YY is used, then YY indicates the year in the 20th century (for example, 31-DEC-92 is December 31, 1992). If you want to indicate years in any century other than the 20th century, then use adifferent format mask, as shown above.

Time Format

Time is stored in 24-hour format #HH:MM:SS. By default, the time in a date field is 12:00:00 A.M. (midnight) ifno time portion is entered. In a time-only entry, the date portion defaults to the first day of the current month. Toenter the time portion of a date, use the TO_DATE function with a formatmask indicating the time portion, as in:

INSERT INTO Birthdays_tab (bname, bday) VALUES ('ANNIE',TO_DATE('13-NOV-92 10:56A.M.','DD-MON-YY HH:MI A.M.'));

To compare dates that have time data, use the SQL function TRUNC if you want to ignore the time component.

Use the SQL function SYSDATE to return the system date and time. The FIXED_DATE initialization parameterallows you to set SYSDATE to a constant; this can be useful for testing.

Experiment No: 2

Creation, modification, configuration, and deletion of databases Commands

AIM:

Creation of a database and tables using DDL commands

COMMANDS

Create Database

mysql> create database testdb;

Query OK, 1 row affected (0.01 sec)

Use Database created

mysql> use testdb;

Database changed

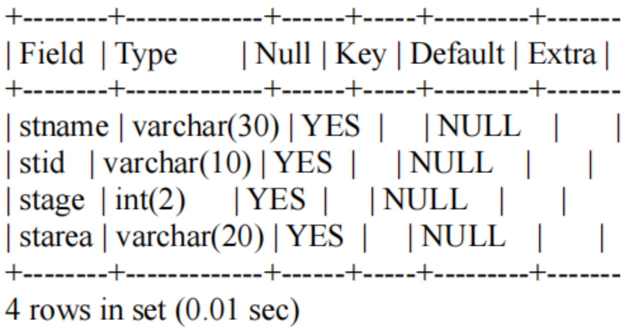
Create Table

create table student (stname varchar(30), stid varchar(10), stage int(2), starea varchar(20));

Query OK, 0 rows affected (0.34 sec)

Description of student

desc student;



```
+--------+-------------+------+-----+---------+-------
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------
| stname | varchar(30) | YES  |     | NULL    |       |
| stid   | varchar(10) | YES  |     | NULL    |       |
| stage  | int(2)      | YES  |     | NULL    |       |
| starea | varchar(20) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------
4 rows in set (0.01 sec)
```

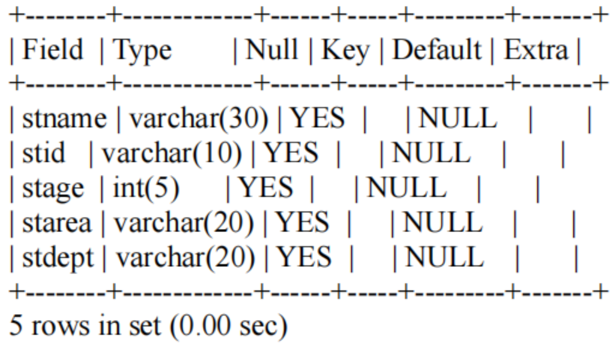
MODIFY TABLE DESCRIPTION

alter table student modify stage int(5);

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

desc student;



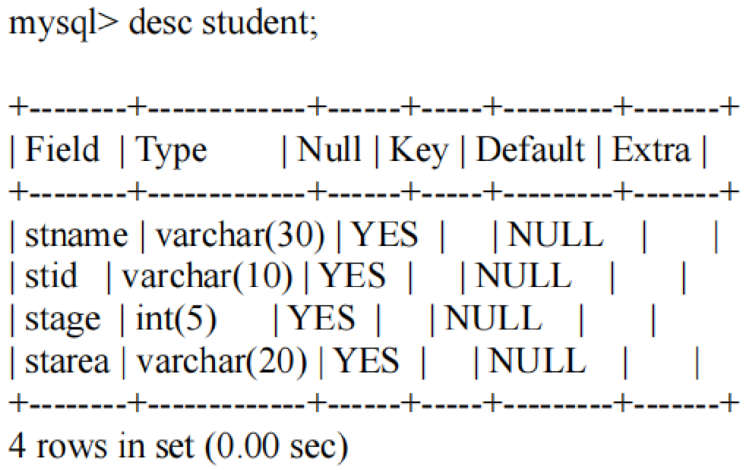
```
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| stname | varchar(30) | YES  |     | NULL    |       |
| stid   | varchar(10) | YES  |     | NULL    |       |
| stage  | int(5)      | YES  |     | NULL    |       |
| starea | varchar(20) | YES  |     | NULL    |       |
| stdept | varchar(20) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

alter table student drop stdept;

Query OK, 0 rows affected (0.55 sec)

Records: 0 Duplicates: 0 Warnings: 0

desc student;



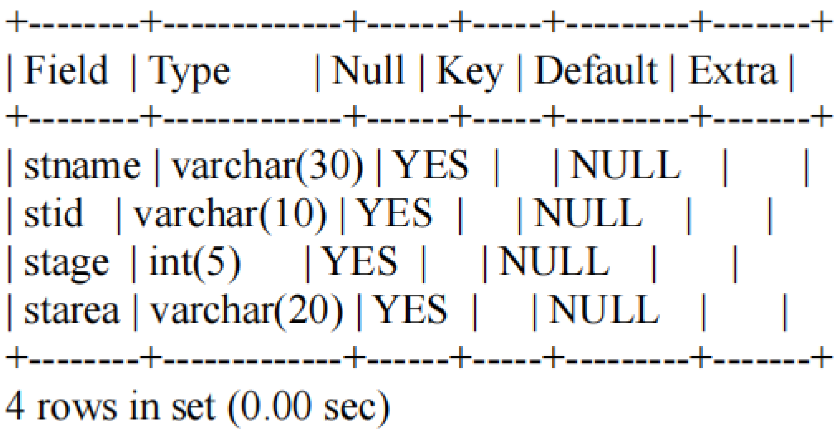
```
mysql> desc student;

+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| stname | varchar(30) | YES  |     | NULL    |       |
| stid   | varchar(10) | YES  |     | NULL    |       |
| stage  | int(5)      | YES  |     | NULL    |       |
| starea | varchar(20) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

CLEAR ALL VALUES IN TABLE

truncate table student;

Query OK, 0 rows affected (0.25 sec)

mysql> desc student;



```
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| stname | varchar(30) | YES  |     | NULL    |       |
| stid   | varchar(10) | YES  |     | NULL    |       |
| stage  | int(5)      | YES  |     | NULL    |       |
| starea | varchar(20) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

DELETE TABLE BOTH SCHEMA AND DATA

```
 drop table student;

Query OK, 0 rows affected (0.18 sec)

mysql> desc student;

ERROR 1146 (42S02): Table 'testdb.student' doesn't exist
```

DELETE DATABASE

```
mysql> DROP DATABASE databasename;

Database dropped
```

RESULT

Query has run successfully and result is obtained.

By constructing queries using SQL I was able to identify the queries for dealing with database activities.

Ex. No : 3

Export ER diagram from the database and verify relationships

AIM

Creation of database schema - DDL (create tables, set constraints, enforce relationships, create indices, delete and modify tables). Export ER diagram from the database and verify relationships**

Consider the employee database given below

emp (emp_id,emp_name, Street_No, city)
works (emp_id, company name, salary)
company (company name, city)
manages (emp_id, manager_id)

Note: Emp_id should start with 'E' in Emp table and emp_id in works table must be the emp_id from emp table . emp_id and manager_id in manages table must be the emp_id from emp table

I. Add these four tables with sufficient constraints.

II. Alter table emp add a constraint that emp_name cannot be null

III. Export ER diagram from database and verify relationships.

COMMANDS

I. A)Create table emp

```
Create table emp(emp_id char(8) check(emp_id like 'E%') primary key, emp_name varchar(18),street_no int,city varchar(18));
```

B) Create table company

```
Create table company(company_name varchar(18) primary key, city varchar(18));
```

C) Create table works

```
Create table works(emp_id char(8) references emp(emp_id),company_name varchar(18) references company(company_name),salary float,primary key(emp_id,company_name));
```

D) Create table manages

```
Create table manages(emp_id char(8) references emp2(emp_id),manager_id char(8) references emp2(emp_id),unique(emp_id,manager_id));
```

II. Alter table emp

```
alter table emp MODIFY emp_name varchar(18) NOT NULL;
```

III. Export ER Diagram

RESULT

Query has run successfully and result is obtained.

By constructing queries using SQL I was able to identify the queries for dealing with database activities.

Ex. No : 4

.

Database initialization - Data insert, Data import to a database (bulk import using UI and SQL Commands)**.

AIM

To insert data to tables used in experiment no 3 using insert commands and bulk import using UI and sql commands.

COMMANDS

1. INSERT COMMANDS

```
insert into emp values(‘E-101’,'Adarsh',101,'MG Road');
insert into emp values(‘E-102’,'Bonny',101, 'MG Road');
insert into emp values(‘E-103’,'Catherine', 102, 'Cochin');
insert into emp values(‘E-104’,'Glenn', 104, 'Ernakulam');
insert into emp values(‘E-105’,'George',  201,'MG Road');
insert into emp values(‘E-106’,'Hayes', 101, 'MG Road');
insert into emp values(‘E-107’,'Johnson',102,’Cochin’);
insert into emp values(‘E-108’,'Jones',  101, 'Cochin');
insert into emp values(‘E-109’,‘Karthik’,  101, 'Ernakulam');
insert into emp values(‘E-110’,'Lavanya',  101, 'Palace Road');
insert into emp values(‘E-111’,'Niharika',  102, ‘Ernakulam’);


insert into company values('SBI',    'MG Road');
insert into company values('SBT',    'MG Road' );
insert into company values('Federal',’Broadway’);
insert into company values('Indian Bank', ‘Cochin’);
insert into company values('SIB', 'Ernakulam');
insert into company values('HDFC',  'Palace Road');
insert into company values('Axis',’Cochin’);
insert into company values('City bank', 'Ernakulam');


insert into works values('E-101',    'SBI',     71000);
insert into works values('E-102',    'SBI',     90000);
insert into works values('E-103',      'SBT',     40000);
insert into works values(‘E-104’, 'Federal',          37000);
insert into works values('E-105', 'SBT',   17000);
insert into works values('E-106',      'Indian Bank',   30000);
insert into works values('E-107',      ‘SIB’,   21000);
insert into works values(‘E-108', 'SIB',   18000);
insert into works values(‘E-109’, 'Indian Bank',   28000);
```

```
    insert into works values(‘E-110’, 'SBT',  250000);
    insert into works values(‘E-111’, 'Federal',        40000);


    insert into manages values('E-101', 'E-102');
    insert into manages values('E-102', Null);
    insert into manages values('E-103', 'E-110');
    insert into manages values(‘E-104’, 'E-111');
    insert into manages values('E-105', 'E-110');
    insert into manages values('E-106', ‘E-109’);
    insert into manages values('E-107', Null);
    insert into manages values(‘E-108', Null);
    insert into manages values(‘E-109’,Null);
    insert into manages values('E-110', Null);
    insert into manages values(‘E-111’, null);
```

Export table values to a text file

First see where is the path set for secure_file_priv, we can do export and import in this location only(else need to configure it) so use following command:

```
mysql>SHOW VARIABLES LIKE 'secure_file_priv';

+------------------+----------------------+

| Variable_name    | Value                |

+------------------+----------------------+|

secure_file_priv | /var/lib/mysql-files/

|+------------------+----------------------+|


1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM WORKS INTO OUTFILE "/var/lib/mysql-files/out2.txt";

Query OK, 0 rows affected (0.06 sec)
```

To show the contents of the file to which data is exported use cat

```
virgo@virgo-Vostro-230:~$ sudo cat "/var/lib/mysql-files/out2.txt"
```

```
e101	sbt	1000
```

Load values from a text file to SQL Table

```
mysql>  LOAD DATA INFILE "/var/lib/mysql-files/out2.txt" INTO TABLE EMP4.WORKS;
Query OK, 1 row affected (0.05 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

RESULT

Query has run successfully and result is obtained.

By constructing queries using SQL I was able to identify the queries for dealing with database activities.

Ex. No : 5

Practice SQL commands for DML (insertion, updating, altering, deletion of data, and viewing/querying records based on condition in databases)

AIM

Consider the employee database created in Find results for the following questions
a. Find the names of all employees who work for SBI.
b. Find all employees in the database who live in the same cities as the companies for which they work.
c. Find all employees and their managers in the database who live in the same cities and on the same street number as do their managers.
d. Find all employees who earn more than the average salary of all employees of their company.
e. Find the company that pay least total salary along with the salary paid.
f. Give all managers of SBI a 10 percent raise.
g. Find the company that has the most employees
h. Find those companies whose employees earn a higher salary, on average than the average salary at Indian Bank.
i. Query to find name and salary of all employees who earn more than each employee of ‘Indian Bank’

COMMANDS

a) Find the names of all employees who work for SBI.

```
SELECT emp_name FROM works,emp WHERE company_name=’SBI’
and emp.emp_id=works.emp_id;

EMP_NAME
------------------
Adarsh
```

b)Find all employees in the database who live in the same cities as the companies for which they work.

```
SELECT emp.emp_name FROM emp, works,company WHERE
emp.emp_id = works. emp_id AND works. company_name=
company.company_name AND emp.city = company.city

EMP_NAME
------------------
Adarsh
George
```

c)Find all employees and their managers in the database who live in the same cities and on the same street number as do their managers.

```
 SELECT emp.emp_name,e2.emp_name “manager name” FROM emp,emp e2,
manages WHERE emp.emp_id = manages.emp_id AND e2.Emp_id=
 manages.manager_id  AND emp.street_no = e2.street_no AND emp.city = e2.city
```

```
EMP_NAME          manager name
------------------  ------------------
Adarsh              Bonny
```

d) Find all employees who earn more than the average salary of all employees of their company.

```
SELECT emp_name,emp.emp_id,salary FROM works ,emp WHERE salary >
(SELECT AVG (salary) FROM works S WHERE works.company_name
 =S.company_name) and emp.emp_id=works.emp_id
```

```
EMP_NAME            EMP_ID       SALARY
------------------  --------     -----------
Bonny                E-102         90000
Hayes               E-106          30000
Johnson             E-107          21000
Lavanya             E-110         250000
Niharika            E-111          40000
```

e). Find the company that pay least total salary along with the salary paid.

```
SELECT company_name,sum(salary) “SALARY PAID” from Works GROUP
BY company_name HAVING sum(salary) <= all (SELECT sum(salary) FROM
Works GROUP BY company_name)
```

```
COMPANY_NAME     SALARY PAID
  ------------------     ------------------
           SIB                     39000
```

f.) Give all managers of SBI a 10 percent raise.

```
UPDATE works SET salary = salary * 1.1 WHERE emp_id in (select manager_id
from manages) and company_name =’SBT’;
```

g). Find the company that has the most employees

```
SELECT company_name FROM works  GROUP BY company_name
HAVING COUNT (DISTINCT emp_id) >= ALL (SELECT COUNT (DISTINCT
emp_id) FROM works GROUP BY company_name)
```

```
COMPANY_NAME
------------------
SBT
```

h) Find those companies whose employees earn a higher salary, on average than the average salary at Indian Bank.

```
SELECT company_name FROM works GROUP BY company_name HAVING
AVG(salary)> (SELECT AVG(salary) FROM  works WHERE company_name =
‘Indian Bank’ GROUP BY
company_name)
```

```
COMPANY_NAME
------------------
SBI
Federal
SBT
```

i).Query to find name and salary of all employees who earn more than each employee of ‘Indian Bank’

```
SELECT emp_name,salary  FROM works,emp
WHERE salary > (SELECT MAX(salary)  FROM works WHERE company_name =
‘Indian Bank’ GROUP BY company_name) and emp.emp_id=works.emp_id;
```

```
EMP_NAME             SALARY
------------------   ----------
Adarsh                 71000
Bonny                  99000
Catherine              40000
Glenn                  37000
Lavanya               250000
Niharika               40000
```

Ex. No : 6

Implementation of built-in functions in RDBMS

AIM

RDBMS Built in Functions

There are two types of functions:

1) SingleRowFunctions:SingleroworScalarfunctionsreturnavalueforeveryrowthatis processed in a query.

2) GroupFunctions:Thesefunctionsgrouptherowsofdatabasedonthevaluesreturnedbythe query.ThisisdiscussedinSQLGROUPFunctions.Thegroupfunctionsareusedtocalculateaggregatevaluesliketotaloraverage,whichreturnjustonetotaloroneaveragevalueafter processing a group of rows.

There are four types of single row functions.They are:

1) Numeric Functions:These are functions that accept numeric input and return numeric values.

2) CharacterorTextFunctions:Thesearefunctionsthatacceptcharacterinputandcanreturnboth character and number values.

3) DateFunctions:ThesearefunctionsthattakevaluesthatareofdatatypeDATEasinputandreturnvaluesofdatatypeDATE,exceptfortheMONTHS_BETWEENfunction,whichreturnsa number.

4) ConversionFunctions:Thesearefunctionsthathelpustoconvertavalueinoneformto anotherform.ForExample:anullvalueintoanactualvalue,oravaluefromonedatatypeto another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE etc.

Mathematical Functions

```
SQL> select ABS(-100) from dual;
ABS(-100)
----------
100
SQL> select ABS(-6) from dual;
ABS(-6)
----------
6
SQL> select FLOOR(2345.78) FROM DUAL;
FLOOR(2345.78)
--------------
```

```
 2345
SQL> SELECT GREATEST(23,67,90,123,78,50) FROM DUAL;
GREATEST(23,67,90,123,78,50)
----------------------------
123
SQL> SELECT LEAST(34, 21,67,11,89,9) FROM DUAL;
LEAST(34,21,67,11,89,9)
 9
SQL> SELECT LENGTH('RAJESHWARI') FROM DUAL;
LENGTH('RAJESHWARI')
--------------------
10
SQL> SELECT LENGTH(17245637) FROM DUAL;
LENGTH(17245637)
----------------
8
SQL> SELECT SQRT(16) FROM DUAL;
SQRT(16)
----------
4
SQL> SELECT SQRT(99) FROM DUAL;
SQRT(99)
 9.94987437
SQL> SELECT POWER(2,4) FROM DUAL;
POWER(2,4)
----------
16
SQL> SELECT POWER(2,10) FROM DUAL;
POWER(2,10)
-----------
1024
SQL> SELECT power(2,10) FROM DUAL;
POWER(2,10)
-----------
1024
SQL> SELECT ROUND(5.86) FROM DUAL;
ROUND(5.86)
-----------
6
SQL> SELECT ROUND(1001.6) FROM DUAL;
ROUND(1001.6)
-------------
1002
SQL> SELECT ROUND(1001.3) FROM DUAL;
ROUND(1001.3)
-------------
1001
```

```
SQL> SELECT SIN(90) FROM DUAL;
SIN(90)
----------
.893996664
SQL> SELECT COS(45) FROM DUAL;
COS(45)
----------
.525321989
SQL> SELECT TAN(30) FROM DUAL;
TAN(30)
----------
-6.4053312
SQL> SELECT TAN(90) FROM DUAL;
TAN(90)
----------
-1.9952004
SQL> SELECT TAN(180) FROM DUAL;
TAN(180)
----------
1.33869021
SQL> SELECT SIGN(-128) FROM DUAL;
SIGN(-128)
----------
-1
SQL> SELECT SIGN(10) FROM DUAL;
SIGN(10)
----------
1
SQL> SELECT SIGN(0) FROM DUAL;
SIGN(0)
----------
0
SQL> SELECT LN(100) FROM DUAL;
LN(100)
----------
4.60517019
SQL> SELECT LN(10) FROM DUAL;
LN(10)
----------
2.30258509
SQL> SELECT LOG(10,100) FROM DUAL;
LOG(10,100)
-----------
2
SQL> SELECT LOG(100,10) FROM DUAL;
LOG(100,10)
-----------
```

```
.5
SQL> SELECT MOD(4,3) FROM DUAL;
MOD(4,3)
----------
1
SQL> SELECT MOD(4,2) FROM DUAL;
MOD(4,2)
----------
0
SQL> SELECT EXP(2) FROM DUAL;
EXP(2)
----------
7.3890561
SQL> SELECT EXP(-2) FROM DUAL;
EXP(-2)
----------
.135335283
SQL> SELECT EXP(0) FROM DUAL;
EXP(0)
----------
1
```

Date Functions

```
SQL> SELECT CURRENT_DATE FROM DUAL;
CURRENT_D
---------
14-AUG-19
SQL> SELECT EXTRACT(YEAR FROM SYSDATE) FROM DUAL;
EXTRACT(YEARFROMSYSDATE)
------------------------
2019
SQL> SELECT EXTRACT(DAY FROM SYSDATE) FROM DUAL;
EXTRACT(DAYFROMSYSDATE)
-----------------------
14
SQL> SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;
EXTRACT(MONTHFROMSYSDATE)
-------------------------
8
SQL> SELECT SYSDATE FROM DUAL;
SYSDATE
---------
AUG-19
```

String Functions

```
SQL> select ascii('t') from dual;
ASCII('T')
----------
116
SQL> select ascii('a') from dual;
ASCII('A')
----------
97
SQL> select ascii('A') from dual;
ASCII('A')
----------
65
SQL>select ascii('Z') from dual;
ASCII('Z')
----------
90
SQL> select ascii('z') from dual;
ASCII('Z')
----------
122
SQL> SELECT UPPER('bldea sb arts and kcp science college') from dual;
UPPER('BLDEASBARTSANDKCPSCIENCECOLLEG’)
-------------------------------------
BLDEA SB ARTS AND KCP SCIENCE COLLEGE
SQL> select LOWER('welcome to dbms lab') from dual;
LOWER('WELCOMETODBM
-------------------
welcome to dbms lab
SQL> select LOWER('WELCOME TO DBMSLAB') from dual;
LOWER('WELCOMETODB
------------------
welcome to dbmslab
SQL> SELECT REPLACE('HELLO','H','K') FROM DUAL;
REPLA
-----
KELLO
SQL> SELECT REPLACE('COMPUTER','C','K') FROM DUAL;
REPLACE( --------
KOMPUTER
SQL> SELECT REPLACE('HELLO','L','A') FROM DUAL;
REPLA
-----
HEAAO
SQL> SELECT TRIM('A' FROM 'ANACONDA') FROM DUAL;
TRIM('
--
```

```
NACOND
SQL> SELECT LTRIM('ANACONDA','A') FROM DUAL;
LTRIM('
-------
NACONDA
SQL> SELECT LTRIM('ANIL','A') FROM DUAL;
LTR
---
NIL
SQL> SELECT RTRIM('ANITA','A') FROM DUAL;
RTRI
---
ANIT
SQL> SELECT RTRIM('ANACONDA','A') FROM DUAL;
RTRIM('
-------
ANACOND
SQL> SELECT RTRIM('ANACONDA ','A') FROM DUAL;
RTRIM('ANAC
-----------
ANACONDA
```

Ex. No : 7

Implementation of various aggregate functions in SQL

AIM

Create the tables with the following fields

Faculty (FacultyCode, FacultyName)
Subject (SubjectCode,SubjectName,MaxMark,FacultyCode)
Student(StudentCode,StudentName,DOB,StudentsBranch(CS/EC/EE/ME), AdmissionDate)
M_Mark (StudentCode, SubjectCode, Mark)

Do the following queries

a) Display the number of faculties.

b) Display the total mark for each student.

c) Display the subject,average mark for each subject.

d) Display the name of subjects for which atleast one student got below 40%.

e) Display the name,subject and percentage of mark who got below 40 %.

f) Display the faculties and alloted subjects for each faculty

g) Display the name of faculties who take more than one subject.

h) Display name,subject,mark, % of mark in ascending order of mark

Commands

```
 Create Table Faculty (F_Code Number Primary Key, F_Name Varchar(15));
        insert into Faculty values(&facultycode,'&facultyname');


        SELECT * FROM Faculty;

                F_CODE              F_NAME
                -----------         --------------------
                     105            Jayakumar
                     104            Sangeetha
                     102            Bindu
                     101            Silgy
                103             Vidhya
```

```
create table Subject (subjectcode varchar(5) primary key not null,subjectname
char(15),maxmark number(5,2),faculty_code int,foreign key(faculty_code) references
Faculty(f_code));
```

```
insert into Subject values('&subjectcode','&subjectname',&maxmark,&facultycode);
```

```
SUBJECTCODE     SUBJECTNAME     MAXMARK  FACULTYCODE
-------------------   -----------------------   ------------------   ----------------------

503                          DBMS             100              105
501                          Maths            150               101
502                          FSA              100              102
504                          OS                75               103
505                           DC              200              104
508                           DBMS lab      1001              103
```

```
create table Student(studentcode varchar(5) primary key not null,studentname
char(15),dob date,studentbranch char(3),adate date,check(studentbranch
in('cs','ec','ee','me')));
```

```
insert into Student values('&studentcode','&studentname','&dob','&studentbranch','&adate');
```

```
Enter value for studentcode: 1
Enter value for studentname: Amitha
Enter value for dob: 12-jan-1987
Enter value for studentbranch: cs
Enter value for adate: 1-jun-2000
old   1: insert into Student
values('&studentcode','&studentname','&dob','&studentbranch','&adate')
new   1: insert into Student values('1','Amitha','12-jan-1987','cs','1-jun-2000')
insert into student values(2,’vaidehi’,’25-dec-88’,’me’,’1-jun-2000’);
insert into student values(3,’varun’,’2-oct-88’,’me’,’2-jun-2000’);
insert into student values(4,’turner’,’5-sep-88’,’ec’,’1-jun-2000’);
insert into student values(5,’vani’,’20-jul-88’,’ee’,’5-jun-2000’);
insert into student values(6,’binu’,’13-aug-88’,’me’,’10-jun-2000’);
insert into student values(7,’chitra’,’14-nov-86’,’me’,’9-jun-1999’);
insert into student values(8,’dona’,’2-dec-91’,’cs’,’2-jun-2000’);
insert into student values(9,’elana’,’5-feb-90’,’cs’,’1-jun-2000’);
insert into student values(10,’fahan’,’20-mar-88’,’ec’,’5-jun-2000’);
insert into student values(11,’ginu’,’13-apr-88’,’ec’,’10-jun-2000’);
insert into student values(12,’hamna’,’14-may-85’,’ee’,’9-jun-1999’);
```

```
create table M_mark(studentcode varchar(5) references
Student(studentcode),subjectcode varchar(5) references Subject(subjectcode),mark
number(5,2),primary key(studentcode,subjectcode));
```

```
insert into M_mark values('&studentcode','&subjectcode',&mark);

insert into M_mark values(1,501,40);
insert into M_mark values(1,502,70);
insert into M_mark values(1,503,50);
insert into M_mark values(1,504,80);
insert into M_mark values(1,505,40);
insert into M_mark values(1,508,70);
insert into M_mark values(2,501,90);
insert into M_mark values(2,502,89);
insert into M_mark values(2,503,77);
insert into M_mark values(2,504,95);
insert into M_mark values(2,505,74);
insert into M_mark values(2,508,98);
insert into M_mark values(3,501,40);
insert into M_mark values(3,502,43);
insert into M_mark values(3,503,40);
insert into M_mark values(3,504,40);
insert into M_mark values(3,505,40);
insert into M_mark values(3,508,35);
insert into M_mark values(4,501,50);
insert into M_mark values(5,501,60);
insert into M_mark values(6,501,67);
insert into M_mark values(7,501,23);
insert into M_mark values(8,501,43);
insert into M_mark values(9,501,42);
insert into M_mark values(10,505,74);
insert into M_mark values(11,508,98);
insert into M_mark values(12,501,40);
insert into M_mark values(5,502,43);
insert into M_mark values(6,503,40);
insert into M_mark values(7,504,40);
insert into M_mark values(8,505,40);
insert into M_mark values(9,508,35);
insert into M_mark values(10,501,50);
insert into M_mark values(11,501,60);
insert into M_mark values(12,503,67);
insert into M_mark values(5,504,23);
insert into M_mark values(6,504,23);
insert into M_mark values(9,504,1);
insert into M_mark values(10,504,1);
insert into M_mark values(6,502,43);
insert into M_mark values(7,505,42);
```

a) Display the number of faculties.

```
select count(*) “No: of Faculties” from faculty;

        No: of Faculties
```

```
----------------
           5
```

b) Display the total mark for each student.

```
select studentname,sum(mark) “Total Mark” from M_mark,Student where
Student.studentcode= M_mark.studentcode group by studentname;
```

```
STUDENTNAME     SUM(MARK)
---------------          ----------
binu     150
hamna                   107
turner                  50
fahan                   124
vaidehi                 523
chitra                  105
Amitha                  350
ginu                    158
varun                   238
vani               126
dona                     83
elana                   77
```

c) Display the subject,average mark for each subject.

```
select subjectname,round(avg(mark),2) “Average mark” from Subject,M_mark where
Subject.subjectcode= M_mark.subjectcode group by subjectname;
```

```
SUBJECTNAME     Average mark
---------------          ------------
DBMS lab                67.2
DC                       51.67
FSA                     57.6
DBMS                    54.8
Maths                   50.42
OS                      55.6
```

d) Display the name of subjects for which atleast one student got below 40%.

```
select subject.subjectname,count(student1.studentname)”NO: OF STUDENTS” from
subject,m_mark,student1 where student1.studentcode= m_mark.studentcode and
m_mark.mark<(40*maxmark)/100 and subject.SubjectCode=m_mark.Subjectcode
group by subject. Subjectname having count(distinct(m_mark.subjectcode))>=1;
```

```
SUBJECTNAME     NO: OF STUDENTS
---------------          ------------------------
DBMS lab                  2
Maths               1
OS                         4
```

e) Display the name,subject and percentage of mark who got below 40 %.

```
select studentname,
subjectname,mark,maxmark,round((m_mark.mark/maxmark)*100,2)"Percentage"
from subject, student1, m_mark where mark<(40*maxmark/100) and subject.
SubjectCode = m_mark. subjectcode and student1.studentcode
=m_mark.studentcode;
```

f) Display the faculties and alloted subjects for each faculty.

```
select Faculty.f_name,Subject.subjectname from Faculty,Subject where
Faculty.F_code=Subject.FACULTYCODE;
```

```
F_NAME            SUBJECTNAME
---------------   ---------------
Vidhya            DBMS lab
Jayakumar         DBMS
Silgy             Maths
Bindu             FSA
Vidhya            OS
Sangeetha         DC
```

g) Display the name of faculties who take more than one subject.

```
Select f_name name from Faculty  where (select count(subjectcode) from Subject
where Subject.facultycode=Faculty.f_code)>1 group by Faculty.f_name;
```

or

```
select Faculty.f_name,count(subject.SubjectCode) “NO OF SUBJECTS” from
Faculty,subject  where (select count(*) from Subject where
Subject.facultycode=Faculty.f_code)>1 and  Subject.facultycode=Faculty.f_code
group by Faculty.f_name;
```

```
F_NAME          NO OF SUBJECTS

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

Vidhya                  2
```

h) Display name,subject,mark, % of mark in ascending order of mark

```
select studentname,subjectname,mark from Student1,Subject,M_mark where
Student1.studentcode=M_mark.studentcode and Subject.subjectcode=
M_mark.subjectcode order by mark;
```

Ex. No : 8

Implementation of Order By, Group By & Having clause

AIM

Create two tables

Dept(Department_Id, Department_Name , Manager_id, Loc)
Emp(Emp_no , Emp_name,Job , Salary , Hiredate,Comm , Depno)

MANAGER_ID is the empno of the employee whom the employee reports to. DEPTNO is a foreign key.Insert these values into department table

1) Display the name and salary for all employees whose salary is not in the range of 5000 and 35000

2) Display the employee name, job ID, and start date of employees hired between February 20, 1990, and May 1, 1998. Order the query in ascending order by start date.

3) list the name and salary of employees who earn between 5,000 and12,000, and are in department 2 or 4. Label the columns Employee and Monthly Salary,respectively.

4)Display the name and hire date of every employee who was hired in 1994.

5). Display the name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

6) Display the name and job title of all employees who do not have a manager.

7). Display the names of all employees where the third letter of the name is an *a.*

8). Display the name of all employees who have an *a* and an *e* in their name.
9). Display the name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to 2,0000, 4000, or 7,000.

10)Write a query that displays the employee's names with the first letter capitalized and all other letters lowercase and the length of the name for all employees whose name starts with *J*, *A*, or *M*. Give each column an appropriate label. Sort the results by the employees' names.

11)For each employee, display the employee's name, and calculate the number of months between today and the date the employee was hired and years worked. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months and year up to the closest whole number.

12). Write a query to display the name, department number, and department name for all employees.

13) Create a query to display the name and hire date of any employee hired after employee Mathew

14) Display the names and hire dates for all employees who were hired before their managers, along with their manager’s names and hire dates. Label the columns Employee, EmpHired, Manager, and Mgr Hired, respectively.

15) Write a query to display the number of people with the same job.

16). Display the manager number and the salary of the lowest paid employee for that manager.Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is less than 6,000. Sort the output in descending order of salary.

17. Write a query to display each department’s name, location, number of employees, and the average salary for all employees in that department. Label the columns Name, Location, Number of People, and Salary, respectively. Round the average salary to two decimal places.

18). Write a query to display the name and hire date of any employee in the same department as amit. Exclude JOHN.

19. Write a query that displays the employee numbers names of all employees who work in a department with any employee whose name contains a *u*.

20)display employee name and department name of all employees that work in a department that has at least 3 employees. Order the list in alphabetical order first by department name, then by employee name.

21. Write a query to list the length of service of the employees (of the form n years and m months).

COMMANDS

```
CREATE TABLE dept(department_id int primary key , department_name VARCHAR(20) NOT NULL , manager_id int, loc varchar(10));
```

```
create table emp(EMP_no int Primary Key,Emp_Name Varchar(10),Job Varchar(10),Hiredate Date,Salary Float,Comm Float,Depno Int References Dept(Department_Id));
```

```
INSERT INTO emp VALUES(1,'Steven', ‘Marketing’,’06-jan-1995’,24000, NULL,2);
INSERT INTO emp VALUES(2,'Neena', 'FI_ACCOUNT', '06-feb-1987’,34000, NULL,1);
INSERT INTO emp VALUES(3,'Lex', ‘FI_MGR', '06-jan-1980’,240000, NULL,1);
INSERT INTO emp VALUES(4,'Alexander', 'Sa_Rep', '06-jun-1987’,20000, NULL,4);
INSERT INTO emp VALUES(5,'Bruce', 'IT_PROG', '06-jul-1990’,24000, NULL,4);
INSERT INTO emp VALUES(6,'David', 'IT_PROG', '06-sep-1991’,22000, NULL,4);
INSERT INTO emp VALUES(7,'vipin', 'IT_PROG', '16-nov-1987’,28000, NULL,4);
INSERT INTO emp VALUES(8,'Diana', 'Pur_Man', '26-jan-1987’,24000, NULL,3);
INSERT INTO emp VALUES(9,'John', 'FI_ACCOUNT', '1-dec-1992’, 24000, NULL,1);
INSERT INTO emp VALUES(10,'Ismael', ‘CLERK’, '29-mar-1994’, 4000, NULL,3);
INSERT INTO emp VALUES(11,'Mathew', ‘CLERK’, '12-oct-1992’, 46000, 200,3);
INSERT INTO emp VALUES(12,'Hayes', ‘Marketing’, '21-apr-1998’,14000, 1000,2);
INSERT INTO emp VALUES(13,'sarun', ‘Marketing’, '18-may-1993’,18000, NULL,2);
INSERT INTO emp VALUES(14,'Henin',‘FI_MGR', '06-aug-1980’,240000, NULL,1);
INSERT INTO emp VALUES(15,'Greesh',‘Clerk', '06-aug-1980’,240000, NULL,5);
```

```
INSERT INTO dept values(1, 'Administration', null, ‘Boston’);
INSERT INTO dept values(2, 'Marketing', null, ‘Boston’);
INSERT INTO dept values(3, 'Purchase', null, ‘perryridge’);
INSERT INTO dept values(4, 'Programming',null, ‘Hudson’);
INSERT INTO dept values(5, 'HR', null, ‘Hudson’);
```

```
Alter table dept add foreign key(manager_id references emp(emp_id));
```

```
Update dept set manager_id=2 where department_id=1;
Update dept set manager_id=1 where department_id=2;
Update dept set manager_id=8 where department_id=3;
Update dept set manager_id=7 where department_id=4;
```

1) Display the name and salary for all employees whose salary is not in the range of 5000 and 35000

```
SELECT emp_name, salary FROM emp WHERE salary NOT BETWEEN 5000 AND 35000;
```

```
EMP_NAME        SALARY
----------      ----------
Lex             240000
Ismael            4000
Mathew     46000
Henin           240000
```

2) Display the employee name, job ID, and start date of employees hired between February 20, 1990, and May 1, 1998. Order the query in ascending order by start date.

```
SELECT emp_name, job, hiredate FROM emp WHERE hiredate BETWEEN '20-Feb-1990' AND '01-May-1998' ORDER BY hiredate
```

```
EMP_NAME         JOB             HIREDATE
==========       ==========      =========
Bruce             IT_PROG         06-JUL-90
David            IT_PROG         06-SEP-91
Mathew           CLERK           12-OCT-92
John             FI_ACCOUNT      01-DEC-92
Steven           Marketing       18-MAY-93
```

```
Ismael          CLERK           29-MAR-94
Hayes           Marketing       21-APR-98
```

3) list the name and salary of employees who earn between 5,000 and12,000, and are in department 2 or 4. Label the columns Employee and Monthly Salary,respectively.

```
SELECT emp_name "Employee", salary "Monthly Salary" ,depno FROM emp
WHERE salary BETWEEN 5000 AND 30000 AND depno IN (2, 4);
```

```
Employee        Monthly Salary
==========      ==============
Alexander       20000
Bruce           24000
vipin            28000
Hayes           14000
Steven          18000
David           22000
```

4)Display the name and hire date of every employee who was hired in 1994.

```
SELECT emp_name, hiredate FROM emp WHERE hiredate LIKE '%94';
```

```
EMP_NAME        HIREDATE
==========      =========
Ismael          29-MAR-94
```

5). Display the name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

```
SELECT emp_name, salary, comm FROM emp WHERE comm >0
ORDER BY salary DESC, comm DESC;
```

Or

```
SELECT emp_name, salary, comm FROM emp WHERE comm IS NOT NULL
ORDER BY salary DESC, comm DESC;
```

```
EMP_NAME         SALARY          COMM

==========       ==========     ==========

Mathew          46000           200

Hayes            14000          1000
```

6) Display the name and job title of all employees who do not have a manager.

```
SELECT emp_name, job FROM emp,dept  WHERE manager_id IS NULL and
emp.depno=dept.department_id;
```

```
EMP_NAME         JOB
==========       ==========
Greesh                Clerk
```

7). Display the names of all employees where the third letter of the name is an *a.*

```
SELECT emp_name FROM emp WHERE emp_name LIKE '__a%';
```

```
EMP_NAME
==========
Diana
```

8). Display the name of all employees who have an *a* and an *e* in their name.

```
SELECT emp_name FROM emp WHERE emp_name LIKE '%a%' AND emp_name
LIKE '%e%';
```

```
EMP_NAME
==========
Neena
Alexander
Ismael
Mathew
Hayes
```

9). Display the name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to 2,0000, 4000, or 7,000.

```
SELECT emp_name, job, salary FROM emp WHERE job IN ('Sa_rep', 'CLERK')
AND salary NOT IN (2000, 4000, 7000);
```

```
EMP_NAME           JOB              SALARY
==========        ==========        ==========
Alexander         Sa_rep            20000
Mathew            CLERK             46000
```

10)Write a query that displays the employee's names with the first letter capitalized and all other letters lowercase and the length of the name for all employees whose name starts with *J*, *A*, or *M*. Give each column an appropriate label. Sort the results by the employees' names.

```
SELECT INITCAP(emp_name) "Name", LENGTH(emp_name) "Length" FROM emp
WHERE emp_name LIKE 'J%' OR emp_name LIKE 'M%' OR emp_name LIKE
'A%'ORDER BY emp_name;
```

```
Name                    Length
```

```
----------------------------       ----------
Alexander                           9
John                                4
Mathew                              6
```

11)For each employee, display the employee's name, and calculate the number of months between today and the date the employee was hired and years worked. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months and year up to the closest whole number.

.

SELECT emp_name**, ROUND(MONTHS_BETWEEN(SYSDATE, hiredate))** MONTHS_WORKED**, round(MONTHS_BETWEEN(SYSDATE, hiredate)/12,2) "NO: Of YEARS" FROM** emp **ORDER BY MONTHS_BETWEEN(SYSDATE, hiredate);**

12). Write a query to display the name, department number, and department name for all employees.

SELECT emp.emp_name, emp.depno, dept.department_name FROM emp , dept WHERE emp.depno = dept.department_id order by dept.department_name;

13) Create a query to display the name and hire date of any employee hired after employee Mathew

SELECT emp_Name, HireDate **FROM** Emp **WHERE** ((HireDate)>**any(SELECT** HireDate **FROM** Emp **WHERE** emp_Name='Mathew'**));**

```
EMP_NAME                HIREDATE
------------------------  ----- ---------
Hayes                    21-APR-98
Ismael                    29-MAR-94
Steven                   18-MAY-93
John                     01-DEC-92
```

14) Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, EmpHired, Manager, and Mgr Hired, respectively.

SELECT emp.emp_name employee , emp.hiredate "EMP HIRE DATE", emp.salary, manager.emp_name manager, manager.hiredate "MANAGER HIRE DATE" **FROM emp , dept, emp manager WHERE dept.manager_id = manager.emp_no and emp.depno=dept.department_id and**
emp.hiredate < manager.hiredate;

```
EMPLOYEE        EMP HIRE DATE   MANAGER      MANAGER HIRE DATE
-----------------    ------------------------ ----------------    --------------------------------
```

```
Lex             06-JAN-80        Neena           06-FEB-87
Alexander       06-JUN-87        vipin           16-NOV-87
Steven          18-MAY-93        Steven          06-JAN-95
Henin           06-AUG-80        Neena           06-FEB-87
```

15) Write a query to display the number of people with the same job.

```
SELECT job, COUNT(*) “No: of Jobs”FROM emp GROUP BY job;
```

```
JOB            NO: OF JOBS
----------     -------------------
IT_PROG             4
Pur_Man             1
CLERK               2
FI_ACCOUNT          2
FI_MGR              2
Marketing           3
```

16). Display the manager number and the salary of the lowest paid employee for that manager.Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is less than 6,000. Sort the output in descending order of salary.

```
SELECT min(salary) “MINIMUM SALARY”,manager_id, department_name FROM
emp,dept where emp.depno=dept.department_id AND manager_id IS NOT NULL
GROUP BY manager_id, department_name HAVING MIN(salary) > 6000
ORDER BY “MINIMUM SALARY” DESC
```

```
MINIMUM SALARY         MANAGER_ID         DEPARTMENT_NAME
---------------------------   ----------------------   -------------------------------
      24000                   2            Administration
      20000                   7            Programming
      14000                   1            Marketing
```

```
select  emp_name “manager”,emp.depno,emp.emp_no, (select min(salary)  from emp e
where (emp.depno=e.depno) group by e.depno having min(salary)>15000) “salary” from
emp,dept where emp.emp_no=dept. MANAGER_ID          and emp.depno=dept.
DEPARTMENT_ID
```

```
select  emp_name “manager”, (select min(salary)  from emp e where (emp.depno=e.depno) group by
e.depno having min(salary)>13000) “salary” from emp,dept where emp.emp_no=dept.
MANAGER_ID      and emp.depno=dept. DEPARTMENT_ID
```

```
select min(emp.salary)  from emp,emp e where (emp.depno=e.depno) group by e.depno having
min(emp.salary)>15000
```

17. Write a query to display each department's name, location, number of employees, and the average salary for all employees in that department. Label the columns Name, Location, Number of People, and Salary, respectively. Round the average salary to two decimal places.

```
SELECT d.department_name "Name", d.loc "Location ",
COUNT(*) "Number of People", ROUND(AVG(salary),2) "Salary"
FROM emp e, dept d
WHERE e.depno = d.department_id GROUP BY d.department_name, d.loc;
```

```
Name               Location   Number of People    Salary
------------------ ----------   -----------------   ------------
Administration      Boston          4               134500
Marketing           Boston          3               18666.67
Programming         Hudson         4                23500
Purchase           perryridge      3                24666.67
```

18). Write a query to display the name and hire date of any employee in the same department as amit. Exclude JOHN.

```
SELECT emp_name, hiredate FROM emp WHERE depno = (SELECT depno
FROM emp WHERE emp_name = 'John') and emp_name<>'John';
```

```
EMP_NAME                  HIREDATE
----------------------------  --------------
Neena                      06-FEB-87
Lex                        06-JAN-80
Henin                      06-AUG-80
```

19. Write a query that displays the employee numbers names of all employees who work in a department with any employee whose name contains a *u*.

```
SELECT emp_no, emp_name,department_name FROM emp,dept
WHERE depno IN (SELECT depno FROM emp WHERE emp_name like '%u%') and
emp.depno=dept.department_id;
```

```
 EMP_NO    EMP_NAME              DEPARTMENT_NAME

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

      6         David               Programming
      7         vipin                       Programming
      5         Bruce                       Programming
      4         Alexander                   Programming
```

20)display employee name and department name of all employees that work in a department that has at least 3 employees. Order the list in alphabetical order first by department name, then by employee name.

SELECT Emp_name, department_name **FROM** emp, dept **WHERE** emp.depno = dept.department_id **AND** emp.depno in (**SELECT** depno **FROM** emp **GROUP BY** depno **HAVING** count(*) >4) **ORDER BY** department_name, emp_name;

21. Write a query to list the length of service of the employees (of the form n years and m months).

SELECT emp_name "employee",to_char(trunc(months_between(sysdate,hiredate)/12))||' years '|| to_char(trunc(mod(months_between (sysdate, hiredate),12)))||' months ' "length of service" **FROM** emp;

Ex. No : 9

Implementation of set operators nested queries, and join queries

AIM

Consider the schema for MovieDatabase:

ACTOR (Act_id, Act_Name, Act_Gender)

DIRECTOR (Dir_id, Dir_Name, Dir_Phone)

MOVIES (Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id)

MOVIE_CAST (Act_id, Mov_id, Role) RATING (Mov_id, Rev_Stars)

Write SQL queries to

1. List the titles of all movies directed by'Hitchcock'.
2. Find the movie names where one or more actors acted in two or moremovies.
3. List all actors who acted in a movie before 2000 and also in a movieafter 2015 (use JOINoperation).
4. Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.
5. Update rating of all movies directed by 'Steven Spielberg' to5.

Table Creation

CREATE TABLE ACTOR (ACT_ID NUMBER (3), ACT_NAME VARCHAR (20), ACT_GENDER CHAR (1), PRIMARY KEY (ACT_ID));

CREATE TABLE DIRECTOR (DIR_ID NUMBER (3), DIR_NAME VARCHAR (20), DIR_PHONE NUMBER (10), PRIMARY KEY (DIR_ID));

CREATE TABLE MOVIES (MOV_ID NUMBER (4), MOV_TITLE VARCHAR (25), MOV_YEAR NUMBER (4), MOV_LANG VARCHAR (12), DIR_ID NUMBER (3), PRIMARY KEY (MOV_ID), FOREIGN KEY (DIR_ID) REFERENCES DIRECTOR (DIR_ID)); Act_Gender Mov_Year Mov_Lang Dir_id FOREIGN KEY (DIR_ID) REFERENCES DIRECTOR (DIR_ID));

CREATE TABLE MOVIE_CAST (ACT_ID NUMBER (3), MOV_ID NUMBER(4), ROLE VARCHAR(10), PRIMARY KEY (ACT_ID, MOV_ID), FOREIGN KEY (ACT_ID)

REFERENCES ACTOR (ACT_ID), FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));

CREATE TABLE RATING (MOV_ID NUMBER (4), REV_STARS VARCHAR (25), PRIMARY KEY (MOV_ID), FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));

Insertion of Values to Tables

INSERT INTO ACTOR VALUES (301,‘ANUSHKA‘,‘F‘);

INSERT INTO ACTOR VALUES (302,‘PRABHAS‘,‘M‘);

INSERT INTO ACTOR VALUES (303,‘PUNITH‘,‘M‘);

INSERT INTO ACTOR VALUES (304,‘JERMY‘,‘M‘);

INSERT INTO DIRECTOR VALUES (60,‘RAJAMOULI‘, 8751611001);

INSERT INTO DIRECTOR VALUES (61,‘HITCHCOCK‘, 7766138911);

INSERT INTO DIRECTOR VALUES (62,‘FARAN‘, 9986776531);

INSERT INTO DIRECTOR VALUES (63,‘STEVEN SPIELBERG‘, 8989776530);

INSERT INTO MOVIES VALUES (1001,‘BAHUBALI-2‘, 2017, _TELAGU‘, 60); INSERT INTO MOVIES VALUES (1002,‘BAHUBALI-1‘, 2015, _TELAGU‘, 60); INSERT INTO MOVIES VALUES (1003,‘AKASH‘, 2008, _KANNADA‘, 61);

INSERT INTO MOVIES VALUES (1004,‘WAR HORSE‘, 2011, _ENGLISH‘, 63); INSERT INTO MOVIE_CAST VALUES (301, 1002, _HEROINE‘);

INSERT INTO MOVIE_CAST VALUES (301, 1001, _HEROINE‘);

INSERT INTO MOVIE_CAST VALUES (303, 1003, _HERO‘);

INSERT INTO MOVIE_CAST VALUES (303, 1002, _GUEST‘);

INSERT INTO MOVIE_CAST VALUES (304, 1004, _HERO‘);

INSERT INTO RATING VALUES (1001,4);

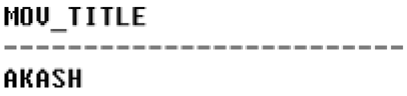
INSERT INTO RATING VALUES (1002,2);

INSERT INTO RATING VALUES (1003, 5);

INSERT INTO RATING VALUES (1004, 4);

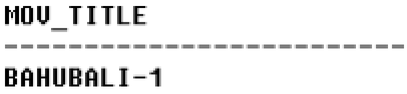
1. List the titles of all movies directed by‘Hitchcock’.

SELECT MOV_TITLE FROM MOVIES WHERE DIR_ID IN (SELECT DIR_ID FROM DIRECTOR WHERE DIR_NAME = _HITCHCOCK‘);



```
MOV_TITLE
-------------------------
AKASH
```

2. Find the movie names where one or more actors acted in two or moremovies.
SELECT MOV_TITLE FROM MOVIES M, MOVIE_CAST MV WHERE M.MOV_ID=MV.MOV_ID AND ACT_ID IN (SELECT ACT_ID FROM MOVIE_CAST GROUP BY ACT_ID HAVING COUNT (ACT_ID)>1) GROUP BY MOV_TITLE HAVING COUNT (*)>1;



```
MOV_TITLE
-------------------------
BAHUBALI-1
```

3. List all actors who acted in a movie before 2000 and also in a movie after 2015 (use JOINoperation).

SELECT ACT_NAME, MOV_TITLE, MOV_YEAR

FROM ACTOR A JOIN

MOVIE_CASTC

ON A.ACT_ID=C.ACT_ID

JOIN MOVIESM

ON C.MOV_ID=M.MOV_ID

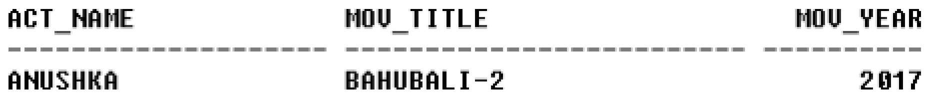
WHERE M.MOV_YEAR NOT BETWEEN 2000 AND 2015; OR

SELECT A.ACT_NAME, A.ACT_NAME, C.MOV_TITLE, C.MOV_YEAR FROM ACTOR A, MOVIE_CAST B, MOVIES C

WHERE A.ACT_ID=B.ACT_ID

AND B.MOV_ID=C.MOV_ID

AND C.MOV_YEAR NOT BETWEEN 2000 AND 2015;



```
ACT_NAME             MOV_TITLE                 MOV_YEAR
-------------------- ------------------------- ----------
ANUSHKA              BAHUBALI-2                      2017
```

4.	Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.

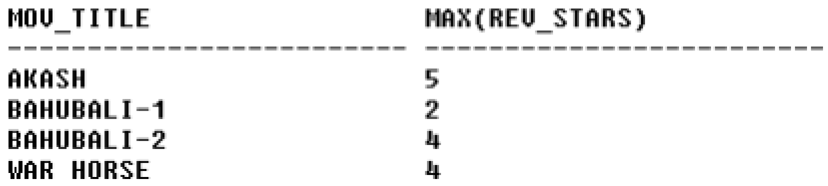
SELECT MOV_TITLE, MAX (REV_STARS)

FROM MOVIES

INNER JOIN RATING USING (MOV_ID)

GROUP BY MOV_TITLE HAVING MAX (REV_STARS)>0

ORDER BY MOV_TITLE;



```
MOV_TITLE                 MAX(REV_STARS)
------------------------- -------------------------
AKASH                     5
BAHUBALI-1                2
BAHUBALI-2                4
WAR HORSE                 4
```

5.	Update rating of all movies directed by ‘Steven Spielberg’ to5 KL

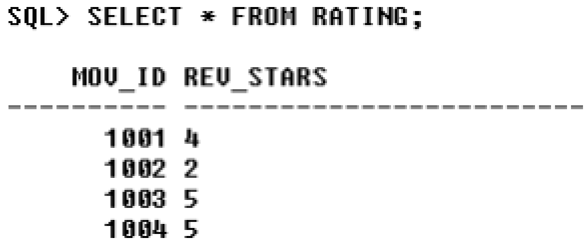
UPDATE RATING

SET REV_STARS=5

WHERE MOV_ID IN

(SELECT MOV_ID FROM MOVIES WHERE DIR_ID IN

(SELECT DIR_ID FROM DIRECTOR WHERE DIR_NAME = _STEVEN SPIELBERG‘));



```
SQL> SELECT * FROM RATING;

    MOV_ID REV_STARS
---------- -------------------------
      1001 4
      1002 2
      1003 5
      1004 5
```

Ex. No : 10

Practice of SQL TCL commands like Rollback, Commit, Savepoint

TRANSATIONAL CONTROL LANGUAGE (T.C.L):

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be mode permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either role back or commit statement.

COMMIT:

This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

Syntax: SQL>COMMIT;

Example: SQL>COMMIT;

SAVE POINT: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

Syntax: SQL>SAVE POINT ID;

Example: SQL>SAVE POINT xyz;

ROLL BACK:

A role back command is used to undo the current transactions. We can role back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are role back.

Syntax:

ROLE BACK(current transaction can be role back)

ROLE BACK to save point ID;

Example:

SQL>ROLE BACK;

SQL>ROLE BACK TO SAVE POINT xyz;

SAVE POINT:

Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

Syntax:

```
SQL> SAVE POINT ID;
```

Example: SQL> SAVE POINT xyz;

Ex. No : 11

Practice of SQL DCL commands for granting and revoking user privileges

PRIVILEGES

A privilege is a right to execute an SQL statement or to access another user's object. In Oracle, there are two types of privileges 🚍

- System Privileges 🚍
- Object Privileges ⌨

System Privileges

are those through which the user can manage the performance of database actions. It is normally granted by DBA to users. Eg: Create Session,Create Table,Create user etc.. ⌨

Object Privileges

allow access to objects or privileges on object, i.e. tables, table columns. tables,views etc..It includes alter,delete,insert,select update etc. (After creating the user, DBA grant specific system privileges to user)

GRANT

The DBA uses the GRANT statement to allocate system privileges to other user.

Syntax:

```
SQL> GRANT privilege [privilege…. … ] TO USER ;
```

```
SQL> Grant succeeded
```

Eg: Grant create session, create table, create view to James;

Object privileges vary from object to object.

An owner has all privilege or specific privileges on object.

```
SQL> GRANT object_priv [(column)] ON object TO user;
```

```
SQL>GRANT select, insert ON emp TO James;
```

```
SQL>GRANT select ,update (e_name,e_address) ON emp TO James;
```

CHANGE PASSWORD:

The DBA creates an account and initializes a password for every user.You can change password by using ALTER USER statement.

Syntax:

```
Alter USER IDENTIFIED BY
```

Eg:

```
ALTER USER James IDENTIFIED BY sam
```

REVOKE

REVOKE statement is used to remove privileges granted to other users.The privileges you specify are revoked from the users.

Syntax:

```
REVOKE [privilege.. …] ON object FROM user
```

Eg: 🗣

```
REVOKE create session,create table from James; 🗣
REVOKE select ,insert ON emp FROM James
```

ROLE

A role is a named group of related privileges that can be granted to user.In other words, role is a predefined collection of previleges that are grouped together,thus privileges are easier to assign user.

```
SQL> Create role custom;

SQL> Grant create table, create view TO custom;

SQL> Grant select, insert ON emp TO custom;
```

Eg: Grant custom to James, Steve;

Ex. No : 12

Practice of SQL commands for creation of views and assertions

Aim

View

Create a table employee with the following fields and create a view which contains the name and salary > 10000 and update the view by changing employees salary to 10. Employee(Name, DA, HRA, TA, Salary)

```
create table employee(name varchar2(10),da number(10), hra number(10), ta number(10),salary number(10));

insert into employee values('&name',&da,&hra,&ta,&salary) ;

select * from employee;

NAME        DA     HRA   TA      SALARY

Anil        1000   2000  1000     15000
arun        1000   3000  1500    20000
anu          500   2000   500      9000
beena        900   2500  1000     11000
remya       1500   1000   2000     10000

create view emp as select emp_name,salary from employee where salary>10000;

EMP_NAME  SALARY
arun              20000
anil              15000
beena             11000


update emp set salary=10; 3 rows updated.

 select * from employee;


NAMEDA     HRA   TA     SALARY

anil   1000  2000  1000  10
arun   1000  3000  1500  10
anu    500   2000  500   9000
beena  900   2500  1000  10
```

remya	1500	1000	2000	10000

Assertions

Create an assertion for the above table to mandate the minimum salary to be at least 10000.

CYCLE 2

Ex. No : 13

Implementation of various control structures like IF-THEN, IF-THEN-ELSE, IF-THEN ELSIF, CASE, WHILE using PL/SQL

Procedural Language/Structured Query Language (PL/SQL) is an extension of SQL.

Basic Syntax of PL/SQL

```
DECLARE

        /* Variables can be declared here */

BEGIN

        /* Executable statements can be written here */

EXCEPTION

        /* Error handlers can be written here. */

END;
```

As we want output of PL/SQL Program on screen, before Starting writing anything type (Only Once per session)

```
        SET SERVEROUTPUT ON
```

CONDITIONAL CONTROL IN PL/SQL:

In PL/SQL, the if statement allows you to control the execution of a block of code. In PL/SQL you can use the IF – THEN – ELSIF – ELSE – END IF statements in code blocks that will allow you to write specific conditions under which a specific block of code will be executed

Ex :- PL/SQL to find addition of two numbers

```
DECLARE

A INTEGER := &A;

B INTEGER := &B;

C INTEGER;

BEGIN

C := A + B;
```

DBMS_OUTPUT.PUT_LINE ('THE SUM IS '||C);

END;

/

Decision making with IF statement :-

The general syntax for the using IF--ELSE statement is

```
IF (TEST_CONDITION) THEN

     SET OF STATEMENTS

ELSE

     SET OF STATEMENTS

END IF;
```

For Nested IF—ELSE Statement we can use IF--ELSIF—ELSE as follows

```
IF (TEST_CONDITION) THEN

     SET OF STATEMENTS

ELSIF (CONDITION)

     SET OF STATEMENTS

END IF;
```

Ex:- **Largest of three numbers.**

This program can be written in number of ways, here are the two different ways to write the program.

Using IFELSE

```
DECLARE

     A NUMBER := &A;

     B NUMBER := &B;

     C NUMBER := &C;

     BIG NUMBER;
```

```
BEGIN
     IF (A > B) THEN
          BIG := A;
     ELSE
          BIG := B;
     END IF;
     IF(BIG < C ) THEN
          DBMS_OUTPUT.PUT_LINE('BIGGEST OF A, B AND C IS ' ||
     C);
     ELSE
          DBMS_OUTPUT.PUT_LINE('BIGGEST OF A, B AND C IS ' ||
     BIG);
     END IF;
END; /
```

Using IF—ELSIF—ELSE

```
DECLARE
     A NUMBER := &A;
     B NUMBER := &B;
     C NUMBER := &C;
BEGIN
     IF (A > B AND A > C) THEN
          DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || A);
     ELSIF (B > C) THEN
          DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || B);
     ELSE
```

```
          DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || C);
     END IF;
END; /
```

CASE statement

```
CASE selector
   WHEN 'value1' THEN S1;
   WHEN 'value2' THEN S2;
   WHEN 'value3' THEN S3;
   ...
   ELSE Sn;  -- default case
END CASE;
```

Example

```
DECLARE
     grade char(1) := 'A';
BEGIN
     CASE grade
          when 'A' then dbms_output.put_line('Excellent');
          when 'B' then dbms_output.put_line('Very good');
          when 'C' then dbms_output.put_line('Well done');
          when 'D' then dbms_output.put_line('You passed');
          when 'F' then dbms_output.put_line('Better try
     again');
     else dbms_output.put_line('No such grade');
     END CASE;
END; /
```

Iterative Control

This is the ability to repeat or skip sections of a code block. A loop repeats a sequence of statements. You have to place the keyword loop before the first statement in the sequence of statements that you want repeated and the keywords end loop immediately after the last statement in the sequence. Once a loop begins to run, it will go on forever. Hence loops are always accompanied by a conditional statement that keeps control on the number of times the loop is executed.

You can build user defined exits from a loop, where required.

THE WHILE LOOP:

```
WHILE<condition>

     LOOP <action>

END LOOP;
```

WRITE A PL/SQL PROGRAM TO FINDS SUM OF DIGITS OF A GIVEN NUMBER.

```
DECLARE

     n      INTEGER;

     temp_sum INTEGER;

     r      INTEGER;

BEGIN

     n := 123456;

     temp_sum := 0;SSS

     WHILE n <> 0 LOOP

          r := MOD(n, 10);

          temp_sum := temp_sum + r;

          n := Trunc(n / 10);

     END LOOP;

     dbms_output.Put_line('sum of digits = '|| temp_sum);
```

```
END;
```

Write a PL/SQL program to grade the student according to the following rules Student(name,rollno,mark1,mark2,mark3)

TOTAL MARKS	**GRADE**
>=250	Distinction
180-250	First Class
120-179	Second Class
80-119	Third Class
<80	Fail

The result should be in the following Format

```
STUDENT NAME:
ROLL NO          :
TOTAL MARKS  :
GRADE            :
```

Create table Stud(rollno int primary key,name char(10),mark1 float,mark2 float,mark3 float);

Insert into stud values(&rollno,’&name’,&mark1,&mark2,&mark3);

```
ROLLNO    NAME        MARK1    MARK2    MARK3
----------   ----------      ---------     ----------    ----------
    1      aparna          80       90        78
    2      amritha         90       92        81
    3      binuja          23       18        20
    4      cathy           49       50        50
    5      danish          60       62        61
    6      fayas           76       62        74
```

```
DECLARE
Name Char(10);
No int;
TOTMARK NUMBER(5,2);
BEGIN
Select rollno,name,(mark1+mark2+mark3) into No,name, TOTMARK from stud where
rollno=&no;
IF TOTMARK >=250 THEN
        DBMS_OUTPUT.PUT_LINE ('---------------------------------------------');
         DBMS_OUTPUT.PUT_LINE(‘ ROLL NO      :'||no);
         DBMS_OUTPUT.PUT_LINE('STUDENT NAME :'|| name);
         DBMS_OUTPUT.PUT_LINE('TOTAL MARK   :'|| TOTMARK);
         DBMS_OUTPUT.PUT_LINE('GRADE        :DISTINCTION’);
        DBMS_OUTPUT.PUT_LINE ('---------------------------------------------');
ELSE IF TOTMARK <250 AND TOTMARK >=180 THEN
        DBMS_OUTPUT.PUT_LINE ('---------------------------------------------');
         DBMS_OUTPUT.PUT_LINE(‘ROLL NO       :'||no);
         DBMS_OUTPUT.PUT_LINE('STUDENT NAME  :'|| name);
```

```
            DBMS_OUTPUT.PUT_LINE('TOTAL MARK    :'|| TOTMARK);
            DBMS_OUTPUT.PUT_LINE('GRADE         :First Class’);
            DBMS_OUTPUT.PUT_LINE ('---------------------------------------------');

ELSE IF TOTMARK <=179 AND TOTMARK >=120 THEN
           DBMS_OUTPUT.PUT_LINE ('---------------------------------------------');
            DBMS_OUTPUT.PUT_LINE(‘ROLL NO      :'||no);
            DBMS_OUTPUT.PUT_LINE('NAME        :'|| name);
            DBMS_OUTPUT.PUT_LINE('TOTAL MARK  :'|| TOTMARK);
            DBMS_OUTPUT.PUT_LINE('GRADE       :SECOND Class’);
            DBMS_OUTPUT.PUT_LINE ('---------------------------------------------');
ELSE

            DBMS_OUTPUT.PUT_LINE ('---------------------------------------------');
           DBMS_OUTPUT.PUT_LINE(‘ROLL NO     :'||no);
            DBMS_OUTPUT.PUT_LINE('NAME       :'|| name);
            DBMS_OUTPUT.PUT_LINE('TOTAL MARK :'|| TOTMARK);
           DBMS_OUTPUT.PUT_LINE('FAILED ');
           DBMS_OUTPUT.PUT_LINE ('---------------------------------------------');
END IF;
END IF;
END IF;
END;
/
```

Ex. No : 14

Creation of Procedures, Triggers and Functions

Trigger

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA.. A database that has a set of associated triggers is called an Active Database.

A trigger description contains three parts:

Event : A change to the database that activates the trigger.

Condition : A query or test that is run when the trigger is activated.

Action : A procedure that is executed when the trigger is activated and its condition is true.

An insert, delete, or update statement could activate a trigger, regardless of which user or application invoked the activating statement; users may not even be aware that a trigger was executed as a side effect of their program.

Procedure in PL/SQL

Procedures are written for doing specific tasks. The general syntax of procedure is

```
CREATE OR REPLACE PROCEDURE (Par_Name1 [IN / OUT/ IN OUT] Par_Type1, ….)
IS (Or we can write AS)

Local declarations;

BEGIN

        PL/SQL Executable statements;

        ..

        ..

        ..

        EXCEPTION

                Exception Handlers;

END<Pro Name> ;
```

Mode of parameters

1. IN Mode :- IN mode is used to pass a value to Procedure/Function. Inside the procedure/function, IN acts as a constant and any attempt to change its value causes compilation error.
2. OUT Mode : The OUT parameter is used to return value to the calling routine. Any attempt to refer to the value of this parameter results in null value.
3. IN OUT Mode : IN OUT parameter is used to pass a value to a subprogram and for getting the updated value from the subprogram.

Function

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

```
CREATE [OR REPLACE] FUNCTION function_name

[(parameter_name [IN | OUT | IN OUT] type [, ...])]

RETURN return_datatype

{IS | AS}

BEGIN

< function_body >

END [function_name];
```

Where,

- function-name specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a return statement.
- The RETURN clause specifies the data type you are going to return from the function.
- function-body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

4.

Ex. No. 14 a) TRIGGER

Aim:

A Library database contain the following tables.

Book_avail (bookid, title, no_of _copies, price)
Student (st_id,name,class,fine)
Issue_tab (st_id, book_id, issuedate, returndate)

Create a database trigger to calculate the fine based on the rules given below.
After 1 month 5% of price
After 2 month 10% of price
After 3 month 20% of price.

INPUT

```
create or replace trigger t2
after update of return on issue
for each row
declare
pr int;
months int;
begin
select price into pr from book where bid=:old.bid;
months:=months_between(:new.return,:old.issuedate);
if months>=1 and months<2 then
update st set fine=pr*0.05 where stid=:old.stid;
else if months>=2 and months<3 then
update st set fine=pr*0.1 where stid=:old.stid;
else if months>=3 then
update st set fine=pr*0.2 where stid=:old.stid;
end if;
end if;
end if;
end;
/
```

PL/SQL

```
set serveroutput on;
declare
id st.stid%type;
dat issue.return%type;
begin
id:=&id;
dat:=&dat;
update issue set return=dat where stid=id;
end;
```

/

Ex. No. 14 b) PROCEDURE

Aim:

Create table **Employee(eno,ename,deptno,salary)**

1. Write a procedure to calculate the income taxpaid as follows.
   a) If gross salary for a financial year is less than 1 lakh, he needs to pay no tax.
   b) If gross salary is between 1 lakh and 1.5 lakh, tax is calculated as 10% of amount exceeding 1 lakh
   c) If gross salary is between 1.5 lakh and 2 lakhs, 20% of the amount exceeding 1 lakh is taxable.
   d) If gross salary is above 2 lakhs, 30% of the amount exceeding 1 lakh is taxable.

Store the details in a new table having fields eno, deptno, & tax_amount

PROCEDURE

```
create or replace procedure inctax(sal IN number,tax OUT number) is
a number(15);
begin
a:=sal*12;
if (a<=100000) then
tax:=0;
else if (a<=150000 and a>100000) then
tax:=((a-100000)*0.1);
else if (a<=200000 and a>150000) then
tax:=((a-100000)*0.2);
else if (a>200000) then
tax:=((a-100000)*0.3);
end if;
end if;
end if;
end if;
end;
/
```

PL/SQL

```
declare
t tax.tax_amount%type;
s em.salary%type;
no em.eno%type;
dno em.deptno%type;
begin
no:=&no;
select salary,deptno into s,dno from em where eno=no;
inctax(s,t);
insert into tax(eno,deptno,tax_amount)values(no,dno,t);
```

```
dbms_output.put_line('Emp No:'||no);
dbms_output.put_line('Dept No:'||dno);
dbms_output.put_line('Tax:'||t);
commit;
end;
/


SQL> select * from em;

       ENO ENAME                    DEPTNO     SALARY
---------- -------------------- ---------- ----------
         1 cyril                       101      12000
         2 jarish                      102      10000
         3 amruth                      101      11000
         4 arun                        105       9000
  5 jeron                  105       8500
         6 akhil                       111        750
         8 faizal                      106      20000
         9 able                        105      19000
        10 abhijith                    103      17500

9 rows selected.

SQL> select * from tax;

       ENO     DEPTNO TAX_AMOUNT
---------- ---------- ----------
         1        101       4400
         2        102       2000
         1        101       4400
         2        102       2000
  1        101       4400
         2        102       2000

6 rows selected.



SQL> /
Enter value for no: 1
old   7: no:=&no;
new   7: no:=1;

PL/SQL procedure successfully completed.

Commit complete.
```

Ex. No. 14 c) FUNCTION

Aim:

Create the following table :
Item (item-code, item-name, qty-in-stock, reorder-level)
Supplier (supplier-code, supplier-name, address)
Can-Supply (supplier-code, item-code)
Write PL/SQL function to do the following:
Set the status of the supplier to "important" if the supplier can supply more than five items.

Program:

```
create or replace function stat(s IN int)
return int is
n int;
begin
select count(itemcode) into n from cansup where supplicode=s;
if n>=5 then
update itm set status='important' where itemcode in(select itemcode from cansup where
supplicode=s);
return(1);
else if n<5 then
return(0);
end if;
end if;
end;
```

PL/SQL

```
set serveroutput on;
declare
cursor c is
select supplicode,suppliname from suppli;
scode suppli.supplicode%type;
sname suppli.suppliname%type;
flag int;
begin
open c;
loop
fetch c into scode,sname;
exit when c%notfound;
flag:=stat(scode);
if flag=1 then
dbms_output.put_line('supplier   '||sname||'   with   '||scode||'    is important');
end if;
end loop;
close c;
commit;
```

```
end;
```