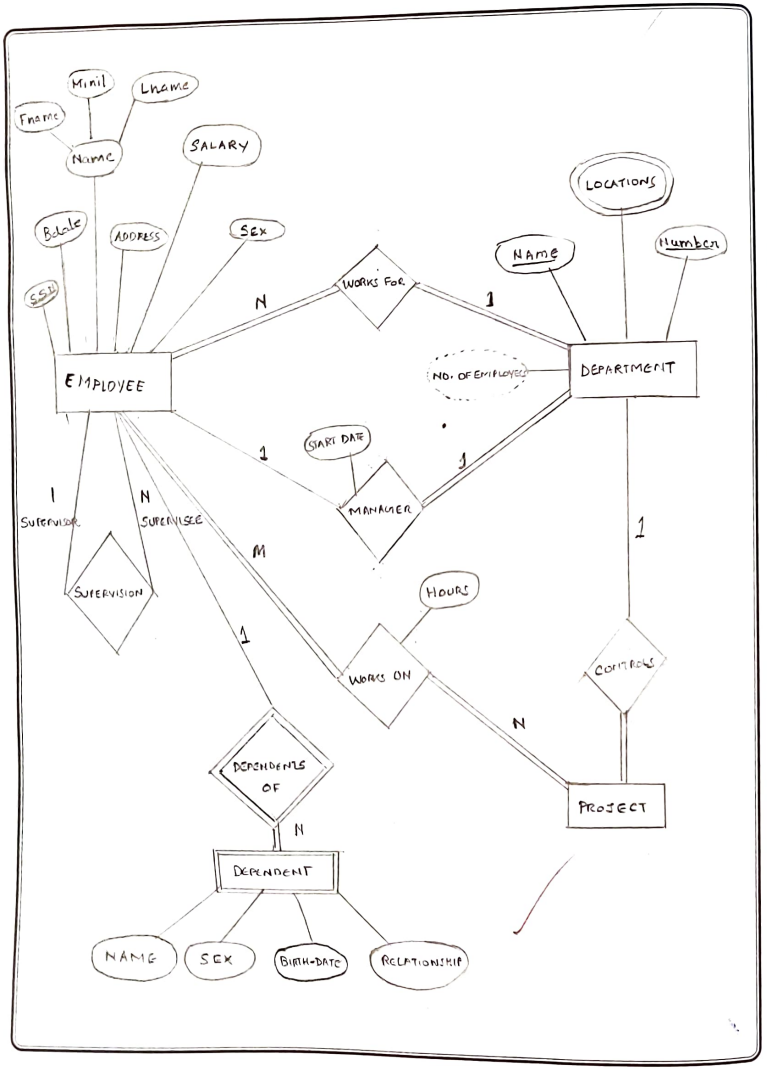
INDEX

Sl. No.	NAME OF EXPERIMENT	DATE	PAGE No.	INITIAL OF TEACHER
1.	ER DIAGRAM	22.11.21	1	
2.	CREATION, MODIFICATION, CONFIGURATION AND DELETION OF DATABASES	25.11.21	5	
3.	DATABASE SCHEMA	29.11.21	7	
4.	EXPORTING ER DIAGRAM	02.12.21	12	
5.	DATABASE INTIALIZATION	06.12.21	14	
6.	SQL COMMANDS FOR DML	09.12.21	15	
7.	BUILT-IN FUNCTIONS IN RDBMS	13.12.21	17	
8.	AGGREGATE FUNCTIONS	16.12.21	19	
9.	ORDER BY, GROUP BY & HAVING	20.12.21	20	
10.	SET OPERATORS, NESTED QUERIES & JOIN QUERIES	03.01.22	22	
11.	TEMP TABLES	06.01.22	24	
12.	SQL TCL COMMANDS	10.01.22	25	
13.	SQL DCL COMMANDS	13.01.22	27	
14.	VIEWS AND ASSERTIONS	17.01.22	28	
15.	CONTROL STRUCTURES	20.01.22	30	
16.	PROCEDURES, TRIGGERS & FUNCTIONS	24.01.22	32	
17.	PACKAGES	27.01.22	34	
18.	CURSORS	31.01.22	35	
19.	EXCEPTION HANDLING	03.02.22	37	
20.	NoSQL DATABASES AND CRUD	07.02.22	39	



22/11/21

EXPERIMENT NO. 1

ER DIAGRAM

AIM:

To design a database schema for an application with ER diagram from a problem description

PROBLEM DESCRIPTION:

Consider a company which is divided into departments. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager.

A department may have several locations and each department controls multiple projects. Each project has a unique name, unique number and it is at a single location.

We store each employee's social security number, address, salary, sex, birthdate. An employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.

Each employee may have a number of dependents. For each dependent we keep track of their name, sex, birthdate and relationship to the employee.

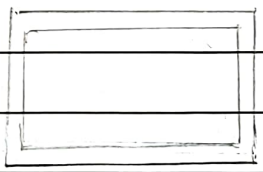
THEORY :

ER diagram is a high level conceptual data model which depicts collection of entities and relationship between them. It is used for conceptual data design of database applications. An entity may be an object with a physical existence like a particular employee, car, house etc or it may be an object with a conceptual existence like company, a job or a university course. Entity is distinguished from other objects on basis of attributes.

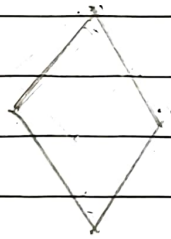
Relationship relates two or more distinct entities with a specific meaning. It is an association between two or more entities of same or different entity set.

The notations used in an ER diagram are as follows :

— Entity

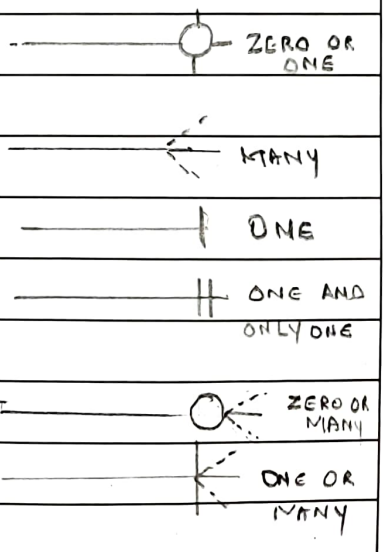


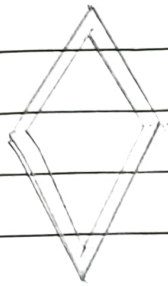
— Weak Entity



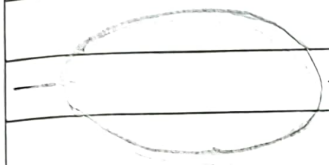
— Relationship

CARDINALITY

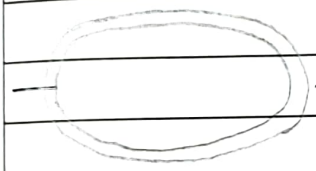




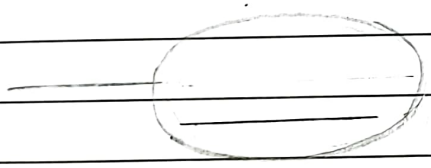
Identifying Relationship



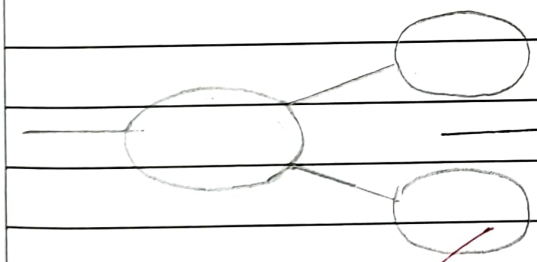
Attribute



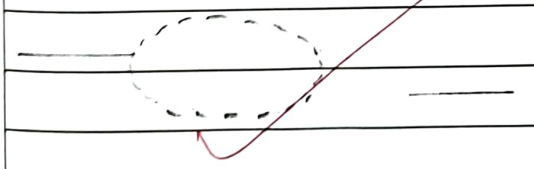
Multivalued Attribute



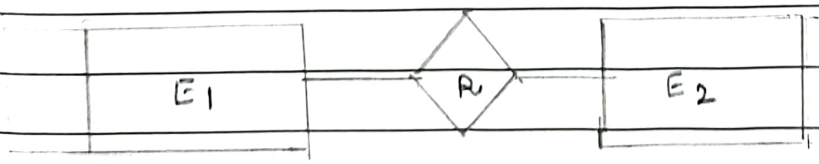
Key Attribute



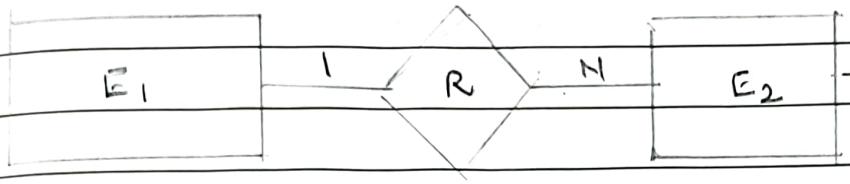
Composite Attribute



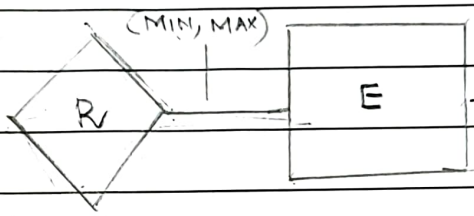
Derived Attribute



Total participation of E_2 in R



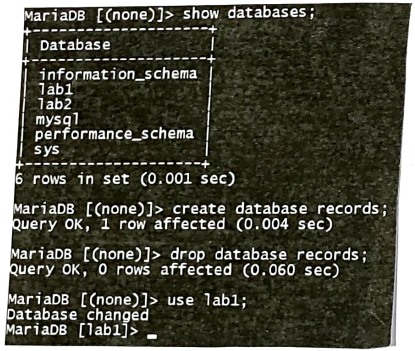
— Cardinality ratio 1:N for $E_1 : E_2$ in R



— Structural constraint on participation of E in R

CONCLUSION :

The ER diagram for the given problem statement has been constructed.



```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| lab1               |
| lab2               |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.001 sec)

MariaDB [(none)]> create database records;
Query OK, 1 row affected (0.004 sec)

MariaDB [(none)]> drop database records;
Query OK, 0 rows affected (0.060 sec)

MariaDB [(none)]> use lab1;
Database changed
MariaDB [lab1]>
```


EXPERIMENT NO. 2

CREATION, MODIFICATION, CONFIGURATION AND DELETION OF DATABASES

AIM :

Creation, modification, configuration and deletion of databases using UI and SQL commands.

THEORY :

Database is a collection of objects such as tables, views, stored procedures, triggers, functions etc.

Create Database - is used to create database in SQL

Syntax : CREATE DATABASE < database-name >

Alter Database : is used to rename database name, change file location and setting etc

Syntax : ALTER DATABASE < database-name >
MODIFY NAME = < new name >

Drop Database : is used to drop or delete a database.

Syntax : DROP DATABASE < database-name >

RESULT :

The queries were successfully executed.

EXPERIMENT NO. 3

DATABASE SCHEMA

AIM :

Creation of database schema - DDL (create tables, set constraints, enforce relationships, create indices, delete and modify tables)

THEORY :

DDL or data definition language actually consists of SQL commands that can be used to define the database schema. Following are the datatypes in MySQL.

Data type	Description
CHARACTER (n)	Character string. Fixed - length n
VARCHAR (n) or CHARACTER VARYING (n)	Character string. Variable length. Maximum length n
BINARY (n)	Binary string. Fixed - length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY (n) or BINARY VARYING (n)	Binary string. Variable length. Maximum length n
INTEGER (P)	Integer numerical (no decimal). Precision P
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19

Data type	Description
DECIMAL (p, s)	Exact numerical, precision p, scale s. Example: decimal (5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC (p, s)	Exact numerical, precision p, scale s (same as DECIMAL)
FLOAT (p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying minimum precision
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month and day values
TIME	Stores hour, minute and second values
TIMESTAMP	Stores year, month, day, hour, minute, second values
INTERVAL	Composed of a no. of integer fields representing a period of time depending on the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

Creating Tables

To create a table, use the SQL command CREATE TABLE

Syntax: CREATE TABLE <TABLE NAME> (<FIELD NAME> <DATATYPE> <[SIZE]>, ...)

```
MariaDB [testdb]> create table student(admno varchar(10) primary key, name varchar(40) not null, age int);
Query OK, 0 rows affected (0.085 sec)

MariaDB [testdb]> alter table student add stream varchar(5);
Query OK, 0 rows affected (0.052 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> alter table student modify stream varchar(6);
Query OK, 0 rows affected (0.052 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> alter table student change column stream course varchar(5);
Query OK, 0 rows affected (0.133 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> alter table student rename to StudentTable;
Query OK, 0 rows affected (0.088 sec)

MariaDB [testdb]> alter table StudentTable drop column course;
Query OK, 0 rows affected (0.034 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [testdb]> select * from StudentTable;
Empty set (0.012 sec)

MariaDB [testdb]> drop table StudentTable;
Query OK, 0 rows affected (0.032 sec)

MariaDB [testdb]>
```


ALTER TABLE :

It is used to add, delete or drop or modify columns in the existing table. It can also add or drop various constraints on the existing table.

1) Add column in table

Syntax : ALTER TABLE table_name ADD new_column_name column_definition [FIRST | AFTER column_name];

2) Add multiple columns

Syntax : ALTER TABLE table_name ADD new_column_name column_definition [FIRST | AFTER column_name],
ADD new_column_name column_definition [FIRST | AFTER column_name],
.... ;

3) Modify columns

Syntax : ALTER TABLE table_name MODIFY column_name column_definition [FIRST | AFTER column_name];

4) Modify multiple columns

Syntax : ALTER TABLE table_name MODIFY column_name column_definition [FIRST | AFTER column_name],
MODIFY column_name column_definition [FIRST | AFTER column_name],
....;

5) DROP COLUMN

Syntax : ALTER TABLE table_name DROP COLUMN column_name ;

6) Rename column

Syntax : ALTER TABLE table_name CHANGE COLUMN old_name new_name column_defition [FIRST | AFTER column_name];

7) Rename table

Syntax : ALTER TABLE table_name RENAME TO new_table_name;

DROP TABLE :

It is used to delete a whole database or just a table. The DROP statement destroys the object like an existing database / table.

Syntax: DROP object object_name ;

TRUNCATE TABLE :

It is used to make the extent of a table for deallocation. The operation quickly removes all data from the table.

Syntax: TRUNCATE TABLE table_name ;

CONSTRAINTS

1. NOT NULL

It restricts a column from having a null value. It enforces a column to have a proper value.

Eg: CREATE TABLE student (sid INT NOT NULL , Name varchar(60), Age INT);

2. UNIQUE

It ensures that a column will have unique values. It will not have duplicate data.

Eg: CREATE TABLE student (sid INT NOT NULL UNIQUE, Name varchar(60), Age INT);

3. PRIMARY KEY

It uniquely identifies each record in a database. Every value must not be null and be unique

Eg: CREATE TABLE Student (sid INT PRIMARY KEY, NAME varchar(60) NOT NULL , Age INT);

4. FOREIGN KEY

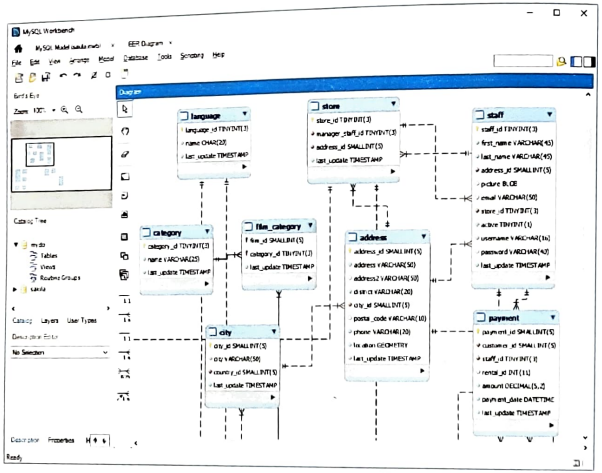
It links two tables together via the primary key. The column of one table points to the primary key of other table.

Eg: CREATE TABLE DEPARTMENT (Dept_name varchar(120) NOT NULL,
sid INT,
FOREIGN KEY (sid) REFERENCES student (sid));

ALTER TABLE DEPARTMENT ADD FOREIGN KEY (sid)
REFERENCES student (sid);

RESULT:

The queries were successfully executed and desired outputs were obtained.



02 / 12 / 21

EXPERIMENT NO. 4

EXPORTING ER DIAGRAM

AIM:

To export ER diagram from the database and verify relationships

THEORY:

An Entity-relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are entity set and relationship set.

An ER diagram shows relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.

In MySQL workbench we reverse engineer a database to obtain ER diagram. Click on database, choose reverse engineer, fill the prompts

and click finish to view ER diagram. We can export it in various formats.

RESULT:

The ER diagram was exported and relationships were verified.

```
MariaDB [mec]> DESC MARK;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| ROLLNO | int(11)     | NO   | PRI | NULL    |       |
| NAME   | varchar(10) | YES  |     | NULL    |       |
| MARK   | int(11)     | YES  |     | NULL    |       |
| GRADE  | varchar(2)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.017 sec)

MariaDB [mec]> SELECT * FROM MARK;
Empty set (0.003 sec)

MariaDB [mec]> LOAD DATA LOCAL INFILE 'c:/Temp/Book1.csv'
    -> INTO TABLE MARK
    -> FIELDS TERMINATED BY ',';
Query OK, 4 rows affected, 3 warnings (0.004 sec)
Records: 4  Deleted: 0  Skipped: 0  Warnings: 3

MariaDB [mec]> SELECT * FROM MARK;
+--------+--------+------+-------+
| ROLLNO | NAME   | MARK | GRADE |
+--------+--------+------+-------+
|      1 | aman   |    9 | A     |
|      2 | varma  |   10 | A+    |
|      3 | gopika |    8 | B+    |
|      4 | kumar  |    7 | B     |
+--------+--------+------+-------+
4 rows in set (0.000 sec)
```

06/12/21

EXPERIMENT NO. 5

DATABASE INITIALIZATION

AIM:

Database initialization - Data insert, Data import to a database

THEORY:

To insert data into the table created INSERT INTO command can be used.

Syntax: INSERT INTO table_name (col1, col2 ..... col N)
VALUES (value1, value2 ..... value N);

We can populate the data into a table through the select statement over another table.

Syntax: INSERT INTO first_table_name [(col1, col2 .... colN)]
SELECT col1, col2 .... colN FROM second_table_name [WHERE condition];

To import data from command line, we can use the following

In maria DB : mysqldump -u username -p database_name > data-dump.sql

In mySQL : LOAD DATA INFILE "filename (with path)"
INTO TABLE tablename;

RESULT:

Data was successfully imported to database

09/12/21

15

EXPERIMENT NO. 6

SQL COMMANDS FOR DML

AIM :

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

THEORY :

Data Manipulation Language (DML) deals with data manipulation and includes most common SQL statements such as SELECT, INSERT, UPDATE, DELETE etc and it stores, modifies, retrieves, deletes and updates data in database.

1. SELECT

It is used to retrieve data from database.

Syntax: SELECT <attribute list> FROM <table list> WHERE <condition>

2. INSERT

It is used to add one or more tuples to a table. Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command.

Syntax: INSERT INTO tablename (col1, col2 ... col N)
VALUES (value1, value2 ... value N),
(value 1, value 2 ... value N), ... ;

```
MariaDB [test]> select * from department;
+---------+-----------+----------+------+----------+-----------+
| dept_id | dept_name | strength | hod  | hod_room | phone     |
+---------+-----------+----------+------+----------+-----------+
|       1 | COMPUTER  |       25 |  101 |      203 | 944264962 |
|       2 | MECHANICAL|       14 |  128 |      306 | 735693462 |
|       3 | CIVIL     |       21 |  304 |      100 | 944658356 |
|       4 | ELECTRONCS|       16 |  403 |      402 | 735683445 |
+---------+-----------+----------+------+----------+-----------+
4 rows in set (0.001 sec)

MariaDB [test]> select dept_name from department
    -> where strength>20;
+-----------+
| dept_name |
+-----------+
| COMPUTER  |
| CIVIL     |
+-----------+
2 rows in set (0.003 sec)

MariaDB [test]> insert into department
    -> values(5,"ELECTRICAL",20,202,104,944782974);
Query OK, 1 row affected (0.005 sec)

MariaDB [test]> update department
    -> set hod=204
    -> where dept_id=4;
Query OK, 1 row affected (0.005 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [test]> delete from department
    -> where dept_name="CIVIL";
Query OK, 1 row affected (0.003 sec)

MariaDB [test]> select * from department;
+---------+-----------+----------+------+----------+-----------+
| dept_id | dept_name | strength | hod  | hod_room | phone     |
+---------+-----------+----------+------+----------+-----------+
|       1 | COMPUTER  |       25 |  101 |      203 | 944264962 |
|       2 | MECHANICAL|       14 |  128 |      306 | 735693462 |
|       4 | ELECTRONCS|       16 |  204 |      402 | 735683445 |
|       5 | ELECTRICAL|       20 |  202 |      104 | 944782974 |
+---------+-----------+----------+------+----------+-----------+
4 rows in set (0.001 sec)
```


3. UPDATE

It is used to update the data of an existing table in database. Single columns as well as multiple columns can be updated using UPDATE statement.

Syntax:

UPDATE table_name SET column1 = value1, column2 = value2.....
WHERE <condition>;

4. DELETE

It is used to delete entries from a database table based on some condition. If no condition is specified, entire records are deleted.

Syntax:

DELETE FROM table_name WHERE <condition>;

RESULT:

The queries were successfully executed and desired outputs were obtained.

```
MariaDB [test]> SELECT ASCII('A'),CHR(68),CONCAT("Hello","World");
+--------------+----------+-------------------------+
| ASCII('A')   | CHR(68)  | CONCAT("Hello","World") |
+--------------+----------+-------------------------+
|           65 | D        | HelloWorld              |
+--------------+----------+-------------------------+
1 row in set (0.002 sec)

MariaDB [test]> SELECT LOWER("TEST"),LENGTH("STRING"),REPLACE("BANANA",'A','O');
+-----------------+-------------------+-----------------------------+
| LOWER("TEST")   | LENGTH("STRING")  | REPLACE("BANANA",'A','O')   |
+-----------------+-------------------+-----------------------------+
| test            |                 6 | BONONO                      |
+-----------------+-------------------+-----------------------------+
1 row in set (0.004 sec)

MariaDB [test]> SELECT UPPER("sample"),TRIM(" Hello World");
+-----------------+-----------------------+
| UPPER("sample") | TRIM(" Hello World")  |
+-----------------+-----------------------+
| SAMPLE          | Hello World           |
+-----------------+-----------------------+
1 row in set (0.000 sec)

MariaDB [test]> SELECT COS(0),LOG10(100),MOD(5,2);
+--------+------------+----------+
| COS(0) | LOG10(100) | MOD(5,2) |
+--------+------------+----------+
|      1 |          2 |        1 |
+--------+------------+----------+
1 row in set (0.004 sec)

MariaDB [test]> SELECT POWER(5,2),SIGN(45),SQRT(49);
+------------+-----------+-----------+
| POWER(5,2) | SIGN(45)  | SQRT(49)  |
+------------+-----------+-----------+
|         25 |         1 |         7 |
+------------+-----------+-----------+
1 row in set (0.001 sec)

MariaDB [test]> SELECT CEIL(4.6),SIN(45),ROUND(3.52,1);
+-----------+--------------------+---------------+
| CEIL(4.6) | SIN(45)            | ROUND(3.52,1) |
+-----------+--------------------+---------------+
|         5 | 0.8509035245341184 |           3.5 |
+-----------+--------------------+---------------+
1 row in set (0.000 sec)
```

13/12/21

17

EXPERIMENT NO.7

BUILT-IN FUNCTIONS IN RDBMS

AIM:

Implementation of built-in functions in RDBMS

THEORY:

The various built-in functions in RDBMS are

1. SQL aggregate functions AVG(), COUNT(), MIN(), MAX() and SUM()
2. SQL String functions

ASCII	Returns ASCII code of the first character of a string
CHR	Returns the character corresponding to input ASCII code
CONCAT	Returns the result of the concatenation of two or more strings
LOWER	Converts all characters in a string to lowercase
LENGTH	Returns the number of characters in a given string. Some database systems use LEN
SUBSTRING	Extracts a substring from a string
REPLACE	Replaces all occurrences of a specified substring in a string by a new substring
TRIM	Removes unwanted characters eg. white spaces from a string
UPPER	Converts all characters in a string to uppercase

3. SQL math functions

ABS	Returns the absolute value
ACOS	Returns the arc cosine of an argument.
ASIN	Returns the arc sine of an argument
ATAN	Returns the arc tangent of an argument.
CEIL, CEILING	Rounds up a float to the nearest integer value

COS	Returns the cosine of an argument
COT	Returns the cotangent of an argument.
EXP	Returns the e constant (2.71828..) that raises to a power of a specified number
LN	Returns the natural logarithm of the argument.
LOG	Returns the natural logarithm of the first argument.
LOG10	Returns the base-10 logarithm of the argument
LOG2	Returns the base-2 logarithm of the argument.
MOD	Returns the remainder (modulo) of a number divided by another.
PI	Returns the value of PI which is 3.14159265358979
POWER	Returns a number raised to a power of specified no.
RAND	Returns a random floating-point value
ROUND	Rounds a number to a specific precision
SIGN	Returns the sign of an argument.
SIN	Returns the sine of an argument.
SQRT	Returns the square root of an argument.
TAN	Returns the tangent of an argument.

4 SQL DATE FUNCTIONS

CURRENT_DATE	Returns current date.
CURRENT-TIME	Returns the current time
CURRENT_TIMESTAMP	Returns current date and time
CAST	Used to convert date to a string
DATEADD	Add an interval to a date
DATEDIFF	Find difference between two dates
DATEPART	Extract part of a date such as year, month and day from given date.

RESULT:

The built-in functions in RDBMS were successfully implemented

```
MariaDB [lab2]> select avg(salary) as Average from employee;
+-------------+
| Average     |
+-------------+
| 99285.7143  |
+-------------+
1 row in set (0.001 sec)

MariaDB [lab2]> select count(name) as Count from employee;
+-------+
| Count |
+-------+
|     7 |
+-------+
1 row in set (0.000 sec)

MariaDB [lab2]> select name,salary as Max from employee where
    -> salary in (select max(salary) from employee);
+------+--------+
| name | Max    |
+------+--------+
| Gopu | 150000 |
+------+--------+
1 row in set (0.071 sec)

MariaDB [lab2]> select name,salary as Min from employee where
    -> salary in (select min(salary) from employee);
+-------+-------+
| name  | Min   |
+-------+-------+
| Tintu | 60000 |
| Maalu | 60000 |
+-------+-------+
2 rows in set (0.001 sec)

MariaDB [lab2]> select sum(salary) as Total from employee;
+--------+
| Total  |
+--------+
| 695000 |
+--------+
1 row in set (0.000 sec)
```

16/12/21

EXPERIMENT NO. 8

AGGREGATE FUNCTIONS

AIM:

Implementation of various aggregate functions in SQL.

THEORY

The following are the SQL aggregate functions.

a) AVG (column_name) - calculate the average of a set of values.

b) Count (column_name) - counts rows in a specified table or view

c) Min (column_name) - Returns the minimum value in a set of values.

d) Max (column_name) - Returns the maximum value in a set of values.

e) Sum (column_name) - calculate the sum of values in a set.

All aggregate functions ignore NULL values except for COUNT function.

RESULT

The queries were successfully executed and desired outputs were obtained.

```
MariaDB [test]> select * from department;
+---------+-------------+----------+------+----------+-----------+
| dept_id | dept_name   | strength | hod  | hod_room | phone     |
+---------+-------------+----------+------+----------+-----------+
|       1 | COMPUTER    |       25 |  101 |      203 | 944264962 |
|       2 | MECHANICAL  |       14 |  128 |      306 | 7356093462 |
|       3 | CIVIL       |       21 |  304 |      100 | 944658356 |
|       4 | ELECTRONICS |       16 |  403 |      402 | 735683445 |
+---------+-------------+----------+------+----------+-----------+
4 rows in set (0.011 sec)

MariaDB [test]> select * from student;
+--------+--------+------+------+-------+-----+
| stu_id | name   | age  | dept | class | sem |
+--------+--------+------+------+-------+-----+
|    301 | AMAL   |   20 |    1 | CSA   |   5 |
|    305 | ADHIL  |   19 |    1 | CSB   |   5 |
|    376 | KIREN  |   19 |    1 | CSA   |   3 |
|    405 | VISHNU |   20 |    2 | MEA   |   5 |
|    515 | AHMAD  |   20 |    3 | CVA   |   5 |
+--------+--------+------+------+-------+-----+
5 rows in set (0.001 sec)

MariaDB [test]> select * from teacher;
+-------+----------+------+---------+----------------+--------+
| tr_id | tr_name  | dept | subject | class_in_charge | salary |
+-------+----------+------+---------+----------------+--------+
|   101 | DEEPA    |    1 | CST104  | CSB            | 103000 |
|   106 | PARVATHY |    1 | CSL101  | NULL           |  85000 |
|   128 | JOHNY    |    2 | MET203  | MEA            | 120000 |
|   217 | VARGHESE |    2 | MEL106  | NULL           |  90000 |
|   304 | RESMI    |    3 | CVT405  | CVA            | 120000 |
+-------+----------+------+---------+----------------+--------+
 rows in set (0.002 sec)
```

20/12/21

EXPERIMENT NO. 9

ORDER BY, GROUP BY & HAVING

AIM:

Implementation of Order By, Group By & Having clause

THEORY:

ORDER BY

This is used to sort the data in ascending or descending order, based on one or more columns. Some database sort the query results in an ascending order by default.

Syntax: SELECT column-list FROM table-name [WHERE condition] [ORDER BY column1, column2 .... columnN] [ASC | DESC];

GROUP BY

This clause is used in collaboration with the SELECT statement to arrange identical data into groups. GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax: SELECT column1, column2 FROM table-name WHERE [conditions] GROUP BY column1, column2 ORDER BY column1, column2

HAVING CLAUSE

This clause enables to specify conditions that filter which group results appear in the results.

```
MariaDB [test]> SELECT * FROM STUDENT
    -> ORDER BY class;
+--------+---------+------+------+-------+------+
| stu_id | name    | age  | dept | class | sem  |
+--------+---------+------+------+-------+------+
|    301 | AMAL    |   20 |    1 | CSA   |    5 |
|    376 | KIREN   |   19 |    1 | CSA   |    3 |
|    305 | AKHIL   |   19 |    1 | CSB   |    5 |
|    515 | ANAND   |   20 |    3 | CVA   |    5 |
|    405 | VISHNU  |   20 |    2 | MEA   |    5 |
+--------+---------+------+------+-------+------+
5 rows in set (0.001 sec)
```

```
MariaDB [test]> SELECT dept,COUNT(*),AVG(salary)
    -> FROM TEACHER
    -> GROUP BY dept;
+------+----------+-------------+
| dept | COUNT(*) | AVG(salary) |
+------+----------+-------------+
|    1 |        2 |  94000.0000 |
|    2 |        2 | 105000.0000 |
|    3 |        1 | 120000.0000 |
+------+----------+-------------+
3 rows in set (0.003 sec)
```

```
MariaDB [test]> SELECT dept,COUNT(dept)
    -> FROM STUDENT
    -> GROUP BY dept
    -> HAVING dept<3;
+------+-------------+
| dept | COUNT(dept) |
+------+-------------+
|    1 |           3 |
|    2 |           1 |
+------+-------------+
2 rows in set (0.000 sec)
```


The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Syntax: SELECT column1, column2 FROM table1, table2
WHERE [conditions]
GROUP BY column1, column2 HAVING [conditions]
ORDER BY column1, column2

RESULT :

The queries were successfully executed and desired outputs were obtained.

```
MariaDB [test]> SELECT tr_name FROM TEACHER
    -> WHERE tr_id IN (SELECT hod FROM DEPARTMENT WHERE dept_id=1);
+---------+
| tr_name |
+---------+
| DEEPA   |
+---------+
1 row in set (0.008 sec)

MariaDB [test]> SELECT name,stu_id
    -> FROM (STUDENT JOIN DEPARTMENT ON dept=dept_id)
    -> WHERE dept_name="COMPUTER";
+-------+--------+
| name  | stu_id |
+-------+--------+
| AMAL  |    301 |
| AKHIL |    305 |
| KIREN |    376 |
+-------+--------+
3 rows in set (0.001 sec)

MariaDB [test]> SELECT tr_name,name
    -> FROM (STUDENT NATURAL JOIN TEACHER)
    -> WHERE class=class_in_charge;
+---------+--------+
| tr_name | name   |
+---------+--------+
| DEEPA   | AKHIL  |
| JOHNY   | VISHNU |
| RESMI   | ANAND  |
+---------+--------+
3 rows in set (0.000 sec)

MariaDB [test]> SELECT hod,tr_name,hod_room
    -> FROM TEACHER CROSS JOIN DEPARTMENT
    -> WHERE tr_id=hod;
+------+---------+----------+
| hod  | tr_name | hod_room |
+------+---------+----------+
|  101 | DEEPA   |      203 |
|  128 | JOHNY   |      306 |
|  304 | RESMI   |      100 |
+------+---------+----------+
3 rows in set (0.001 sec)


MariaDB [test]> SELECT name FROM STUDENT
    -> WHERE class IN (SELECT class_in_charge
    -> FROM TEACHER
    -> WHERE class_in_charge="CSB" AND tr_id=101);
+-------+
| name  |
+-------+
| AKHIL |
+-------+
1 row in set (0.001 sec)
```


EXPERIMENT NO. 10

SET OPERATORS, NESTED QUERIES & JOIN QUERIES

AIM:

Implementation of set operators, nested queries and join queries

THEORY:

Set Operators: They are used to combine the results of two queries.

1) UNION: Combines the results of two SELECT statements. Duplicate rows will be eliminated from the results obtained.

```
SELECT * FROM t_1 UNION SELECT * FROM t_2 ;
```

2) UNION ALL: This operator combines all the records from both the queries. Duplicate rows will not be eliminated from the results obtained after performing this operation.

```
SELECT * FROM t_1 UNION ALL SELECT * FROM t_2 ;
```

3) INTERSECT: It combines two SELECT statements but will return only the records which are common from both SELECT statements.

```
SELECT * FROM t_1 INTERSECT SELECT * FROM t_2 ;
```

4) MINUS: It displays the rows which are present in the first query but absent in the second query with no duplicates

```
SELECT * FROM t_1 MINUS SELECT * FROM t_2;
```