Experiment No. 01

Aim - To identify the case study and detailed statement of the problem,
And to design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model for Library.

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. It is used to model the logical view of the database system.

Components of the ER Diagram:

1. Entity: An Entity may be an object with a physical existence-a particular person, car, house, or employee - or it may be an object with a conceptual existence - a company, a job, or a university course. An Entity is an object of Entity Type and the set of all entities is called an entity set.

2. Attributes: Attributes are the properties which define the entity type. For example, Roll No, Name, DOB, Age, Address, Mobile No are the attributes which define entity type Student. In the ER diagram, the attribute is represented by an ellipse.

Key attribute, Composite attribute, Derived attribute, Multivalued attribute

3. Relationships: A relationship type represents the association between entities

One-to-One Relationship One-to-Many Relationship

May-to-One Relationship Many-to-Many Relationship

Conclusion - Designed an entity relationship diagram for the library.

Experiment No. 02

Aim - To draw an EER Model diagram with specialization, generalization & aggregation.

Theory -

EER stands for Enhanced ER or Extended ER EER Model.

Concepts include all modeling concepts of basic ER.

Additional concepts: subclasses/superclasses, specialization/generalization, categories (UNION types).

The additional EER concepts are used to model applications more completely and more accurately.

Specialization: Specialization is a top-down approach, and it is opposite to Generalization. in specialization, one higher level entity can be broken down into two lower level entities. Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.

Generalization: Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common in generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.In generalization, entities are combined to form a more generalized entity, Le subclasses are combined to make a superclass

Aggregation: In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity

Conclusion - Designed an EER diagram with Specialization, Generalization and Aggregation.

Experiment No. 03

Aim - Write a program to execute all DML operations.

Software Used - MySQL

Theory -

- **INSERT table -**
  To insert a single row into a table, you use the following form of the INSERT statement :
  INSERT INTO table (column1,column2 ,..) VALUES(value1, value2 ,...);

- **UPDATE table -**
  To update existing data in a table, you use SQLite UPDATE statement. The following illustrates the syntax of the UPDATE statement: UPDATE table SET column_1 = new_value_1, column_2 = new_value_2 WHERE search_condition ORDER column_or_expression LIMIT row_count OFFSET offset;

- **DELETE table -**
  The SQLite DELETE statement allows you to delete one row, multiple rows, and all rows in a table. The syntax of the SQLite DELETE statement is as follows:
  DELETE FROM table WHERE search_condition;

Output (ScreenShot) -

- **Insert**

```
mysql> create table students
    -> (roll_no int, name varchar (100), branch varchar (80), age int);
Query OK, 0 rows affected (0.14 sec)

mysql> insert into students
    -> values(51, 'Sudham', 'aiml',20);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students
    -> values(8, 'Prathamesh', 'aiml',20);
Query OK, 1 row affected (0.01 sec)

mysql> insert into students
    -> values(151, 'Monu', 'aiml',19);
Query OK, 1 row affected (0.06 sec)

mysql> insert into students
    -> values(108, 'Babbu', 'aiml',19);
Query OK, 1 row affected (0.01 sec)

mysql> select * from students;
+---------+------------+--------+------+
| roll_no | name       | branch | age  |
+---------+------------+--------+------+
|      51 | Sudham     | aiml   |   20 |
|       8 | Prathamesh | aiml   |   20 |
|     151 | Monu       | aiml   |   19 |
|     108 | Babbu      | aiml   |   19 |
+---------+------------+--------+------+
4 rows in set (0.00 sec)
```

- **Update**

```
mysql> update students
    -> set branch = 'iot' where roll_no = 151;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update students
    -> set branch = 'dse' where roll_no = 108;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from students;
+---------+------------+--------+------+
| roll_no | name       | branch | age  |
+---------+------------+--------+------+
|      51 | Sudham     | aiml   |   20 |
|       8 | Prathamesh | aiml   |   20 |
|     151 | Monu       | iot    |   19 |
|     108 | Babbu      | dse    |   19 |
+---------+------------+--------+------+
4 rows in set (0.00 sec)
```

- **Delete**

```
mysql> select * from students;
+---------+------------+--------+------+
| roll_no | name       | branch | age  |
+---------+------------+--------+------+
|      51 | Sudham     | aiml   |   20 |
|       8 | Prathamesh | aiml   |   20 |
|     151 | Monu       | iot    |   19 |
|     108 | Babbu      | dse    |   19 |
+---------+------------+--------+------+
4 rows in set (0.00 sec)

mysql> delete from students where roll_no = 151;
Query OK, 1 row affected (0.01 sec)

mysql> select * from students;
+---------+------------+--------+------+
| roll_no | name       | branch | age  |
+---------+------------+--------+------+
|      51 | Sudham     | aiml   |   20 |
|       8 | Prathamesh | aiml   |   20 |
|     108 | Babbu      | dse    |   19 |
+---------+------------+--------+------+
3 rows in set (0.00 sec)
```

Conclusion - Executed the DML operations in MySQL.

Experiment No. 04

Aim-Write a program to execute all constraints at row level and table level.

Software Used-MySQL

Theory -

- **NOT NULL Constraint**

  Ensures that a column cannot have a NULL value.

  By default, a column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

- **PRIMARY KEY Constraint**

  A combination of NOT NULL and UNIQUE. Uniquely identifies each row in a table.

  The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

- **CHECK Constraint**

  The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a column it will allow only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

- **FOREIGN KEY**

  In SQLite, FOREIGN KEY Constraint is used to maintain the relationship between multiple tables and it helps us to identify records uniquely Generally, the Foreign Key column in one table becomes a Primary Key constraint in another table to maintain the relationship

between tables. We can create multiple FOREIGN KEY Constraints on columns in the table but all those columns must point to PRIMARY KEY Constraint in other tables. Generally, in SQLite FOREIGN KEY constraint in one table called a child table which points to the PRIMARY KEY constraint of another table is called a parent or reference table.

Syntax:

CREATE TABLE table1 (col1 INTEGER PRIMARY KEY, col2 TEXT NOT NULL); CREATE TABLE table2 (col3 INTEGER PRIMARY KEY, col4 TEXT NOT NULL, col1 INTEGER NOT NULL, FOREIGN KEY (col1) REFERENCES table1);

Output (ScreenShot) -

- **NOT NULL**

```
mysql> alter table students
    -> add gender varchar (10) not null;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from students;
+---------+------------+--------+-----+--------+
| roll_no | name       | branch | age | gender |
+---------+------------+--------+-----+--------+
|      51 | Sudham     | aiml   |  20 |        |
|       8 | Prathamesh | aiml   |  20 |        |
|     108 | Babbu      | dse    |  19 |        |
|     151 | Monu       | aiml   |  19 |        |
+---------+------------+--------+-----+--------+
4 rows in set (0.00 sec)
```

- **PRIMARY KEY**

```
mysql> alter table students
    -> add primary key(roll_no);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from students;
+---------+------------+--------+-----+
| roll_no | name       | branch | age |
+---------+------------+--------+-----+
|       8 | Prathamesh | aiml   |  20 |
|      51 | Sudham     | aiml   |  20 |
|     108 | Babbu      | dse    |  19 |
|     151 | Monu       | aiml   |  19 |
+---------+------------+--------+-----+
4 rows in set (0.00 sec)
```

- **CHECK**

```
mysql> select * from students;
+---------+------------+--------+-----+
| roll_no | name       | branch | age |
+---------+------------+--------+-----+
|       8 | Prathamesh | aiml   |  20 |
|      51 | Sudham     | aiml   |  20 |
|     108 | Babbu      | dse    |  19 |
|     151 | Monu       | aiml   |  19 |
+---------+------------+--------+-----+
4 rows in set (0.00 sec)

mysql> alter table students
    -> add check (age>=19);
Query OK, 4 rows affected (0.08 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

- **FOREIGN KEY**

```
mysql> alter table students
    -> add primary key(roll_no);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from students;
+---------+------------+--------+-----+
| roll_no | name       | branch | age |
+---------+------------+--------+-----+
|       8 | Prathamesh | aiml   |  20 |
|      51 | Sudham     | aiml   |  20 |
|     108 | Babbu      | dse    |  19 |
|     151 | Monu       | aiml   |  19 |
+---------+------------+--------+-----+
4 rows in set (0.00 sec)
```

```
mysql> create table orders
    -> (orderid int not null, ordernumber int not null, primary key (orderid),roll_no int, foreign key(roll_no) references students(roll_no));
Query OK, 0 rows affected (0.06 sec)

mysql> insert into orders
    -> values(234,1,151);
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders
    -> values(235,2,108);
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders
    -> values(236,3,8);
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders
    -> values(237,4,51);
Query OK, 1 row affected (0.01 sec)

mysql> select * from orders;
+---------+-------------+---------+
| orderid | ordernumber | roll_no |
+---------+-------------+---------+
|     234 |           1 |     151 |
|     235 |           2 |     108 |
|     236 |           3 |       8 |
|     237 |           4 |      51 |
+---------+-------------+---------+
4 rows in set (0.00 sec)
```

Conclusion - Executed all constraints at row level and table level.

Experiment No. 05

Aim - Write a program with
a)inner join, b)outer join, c)equi join, d)natural join

Software Used - MySQL

Theory -

- **inner join**
  INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, the column values for each matched pair of rows of A and B are combined into a result row. An INNER JOIN is the most common and default type of join. You can use the INNER keyword optionally.
- **outer join**
  In SQL the FULL OUTER JOIN combines the results of both left and right outer joins and returns all (matched or unmatched) rows from the tables on both sides of the join clause. Because this is a full join, all rows (both matching and nonmatching) from both tables are included in the output. There is only one match between table table_A and table table_B, so only one row of output displays values in all columns. All remaining rows of output contain only values from table table_A or table table_B, with the remaining columns set to missing values.

- **equi join**

  SQL EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables. An equal sign (=) is used as a comparison operator in the where clause to refer to equality.

  Syntax-

  SELECT * FROM table1 JOIN table2 [ON (join_condition)]

- **natural join**

  The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that columns with the same name of associated tables will appear once only.

  Syntax-

  SELECT * FROM table1 NATURAL JOIN table2;

Output (ScreenShot) -

Creation of table -

```
Database changed
mysql> create table orders (orderid int,customerid int,orderdate int);
Query OK, 0 rows affected (0.31 sec)
```

```
mysql> insert into orders (orderid,customerid,orderdate) values (1001,1,15-03-2022);
Query OK, 1 row affected (0.04 sec)

mysql> insert into orders (orderid,customerid,orderdate) values (1002,2,16-03-2022);
Query OK, 1 row affected (0.07 sec)

mysql> insert into orders (orderid,customerid,orderdate) values (1003,3,17-03-2022);
Query OK, 1 row affected (0.04 sec)

mysql> select *from orders;
+---------+------------+-----------+
| orderid | customerid | orderdate |
+---------+------------+-----------+
|    1001 |          1 |     -2010 |
|    1002 |          2 |     -2009 |
|    1003 |          3 |     -2008 |
+---------+------------+-----------+
3 rows in set (0.00 sec)
```

```
mysql> create table customers (customerid int,customername varchar(20),contactnumber int);
Query OK, 0 rows affected (0.20 sec)
```

```
mysql> insert into customers (customerid,customername,contactnumber) values (1,'monu',9372);
Query OK, 1 row affected (0.05 sec)

mysql> insert into customers (customerid,customername,contactnumber) values (2,'prathamesh',9234);
Query OK, 1 row affected (0.04 sec)

mysql> insert into customers (customerid,customername,contactnumber) values (3,'sudham',9278);
Query OK, 1 row affected (0.04 sec)

mysql> insert into customers (customerid,customername,contactnumber) values (4,'xyz',9876);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> select *from customers;
+------------+--------------+---------------+
| customerid | customername | contactnumber |
+------------+--------------+---------------+
|          1 | monu         |          9372 |
|          2 | prathamesh   |          9234 |
|          3 | sudham       |          9278 |
|          4 | xyz          |          9876 |
+------------+--------------+---------------+
4 rows in set (0.00 sec)
```

- **inner join**

```
mysql> select orders.orderid,customers.customername,orders.orderdate
    -> from orders
    -> inner join customers on
    -> orders.customerid=customers.customerid;
+---------+--------------+-----------+
| orderid | customername | orderdate |
+---------+--------------+-----------+
|    1001 | monu         |     -2010 |
|    1002 | prathamesh   |     -2009 |
|    1003 | sudham       |     -2008 |
+---------+--------------+-----------+
3 rows in set (0.05 sec)
```

- **outer join**

```
mysql> select orders.orderid,customers.customername
    -> from orders
    -> left join customers on
    -> orders.customerid=customers.customerid
    -> order by orders.orderid;
+---------+--------------+
| orderid | customername |
+---------+--------------+
|    1001 | monu         |
|    1002 | prathamesh   |
|    1003 | sudham       |
+---------+--------------+
3 rows in set (0.00 sec)
```

```
mysql> select orders.orderid,customers.customername,customers.contactnumber from orders right join customers on orders.customerid=customers.cu
stomerid order by orders.orderid;
+---------+--------------+---------------+
| orderid | customername | contactnumber |
+---------+--------------+---------------+
|    NULL | xyz          |          9876 |
|    1001 | monu         |          9372 |
|    1002 | prathamesh   |          9234 |
|    1003 | sudham       |          9278 |
+---------+--------------+---------------+
4 rows in set (0.00 sec)
```

- **equi join**

```
mysql> select * from orders join customers on orders.customerid=customers.customerid;
+---------+------------+-----------+------------+--------------+---------------+
| orderid | customerid | orderdate | customerid | customername | contactnumber |
+---------+------------+-----------+------------+--------------+---------------+
|    1001 |          1 |     -2010 |          1 | monu         |          9372 |
|    1002 |          2 |     -2009 |          2 | prathamesh   |          9234 |
|    1003 |          3 |     -2008 |          3 | sudham       |          9278 |
+---------+------------+-----------+------------+--------------+---------------+
3 rows in set (0.02 sec)
```

- **natural join**

```
mysql> select *
    -> from orders
    -> natural join customers;
+------------+---------+-----------+--------------+---------------+
| customerid | orderid | orderdate | customername | contactnumber |
+------------+---------+-----------+--------------+---------------+
|          1 |    1001 |     -2010 | monu         |          9372 |
|          2 |    1002 |     -2009 | prathamesh   |          9234 |
|          3 |    1003 |     -2008 | sudham       |          9278 |
+------------+---------+-----------+--------------+---------------+
3 rows in set (0.00 sec)
```

Conclusion - Implemented various join operations.

Experiment No. 06

Aim - Write a program to execute nested queries
a)with aggregate functions, b)with operators, c)string manipulation.

Software Used - MySQL

Theory -

- **aggregate functions**
  >count()- Returns the number of rows that match a specified condition
  >sum()- Sum function is used to calculate the sum of all selected content of columns. It works on numeric fields only.
  >avg()- Returns the average value of a group.
  >max()- Returns the maximum value in a group.
  >min()- returns the minimum value in a group.
- **operators**
  >arithmetic- Arithmetic operators run mathematical operations on two expressions of one or more data types. They're run from the numeric data type category.
  >bitwise- Bitwise operators perform bit manipulations between two expressions of any of the data types of the integer data type category. Bitwise operators convert two integer values to binary bits, perform the AND, OR, or NOT operation on each bit, producing a result. Then converts the result to an integer.
  >logical- Logical operators test for the truth of some condition. Logical operators, like comparison operators, return a Boolean data type with a value of TRUE, FALSE, or UNKNOWN.
- **string manipulation.**
  >length- Return the number of characters in a string or the number of bytes in a BLOB.

>lower- Returns a character expression after converting uppercase character data to lowercase.

>upper- Return a copy of a string with all of the characters converted to uppercase.

>insert- inserts a string within a string at the specified position and for a certain number of characters.

>concat- This function returns a string resulting from the concatenation, or joining, of two or more string values in an end-to-end manner.

Output (ScreenShot) -

- **aggregate functions**

```
mysql> select *from customers;
+------------+--------------+---------------+
| customerid | customername | contactnumber |
+------------+--------------+---------------+
|          1 | monu         |          9372 |
|          2 | prathamesh   |          9234 |
|          3 | sudham       |          9278 |
|          4 | xyz          |          9876 |
+------------+--------------+---------------+
4 rows in set (0.01 sec)
```

```
mysql> select count(*)
    -> from customers;
+----------+
| count(*) |
+----------+
|        4 |
+----------+
1 row in set (0.02 sec)

mysql> select sum(customerid)
    -> from customers;
+-----------------+
| sum(customerid) |
+-----------------+
|              10 |
+-----------------+
1 row in set (0.01 sec)

mysql> select avg(customerid)
    -> from customers;
+-----------------+
| avg(customerid) |
+-----------------+
|          2.5000 |
+-----------------+
1 row in set (0.00 sec)
```

```
mysql> select max(contactnumber)
    -> from customers;
+--------------------+
| max(contactnumber) |
+--------------------+
|               9876 |
+--------------------+
1 row in set (0.00 sec)
```

```
mysql> select min(contactnumber)
    -> from customers;
+--------------------+
| min(contactnumber) |
+--------------------+
|               9234 |
+--------------------+
```

- **operators**

```
mysql> select 5+5 from dual;
+-----+
| 5+5 |
+-----+
|  10 |
+-----+
1 row in set (0.01 sec)

mysql> select 8<51 from dual;
+------+
| 8<51 |
+------+
|    1 |
+------+
1 row in set (0.01 sec)

mysql> select 8>51 from dual;
+------+
| 8>51 |
+------+
|    0 |
+------+
1 row in set (0.00 sec)

mysql> select * from customers;
+------------+--------------+---------------+
| customerid | customername | contactnumber |
+------------+--------------+---------------+
|          1 | monu         |          9372 |
|          2 | prathamesh   |          9234 |
|          3 | sudham       |          9278 |
|          4 | xyz          |          9876 |
+------------+--------------+---------------+
4 rows in set (0.00 sec)

mysql> select *from customers where not customerid>2;
+------------+--------------+---------------+
| customerid | customername | contactnumber |
+------------+--------------+---------------+
|          1 | monu         |          9372 |
|          2 | prathamesh   |          9234 |
+------------+--------------+---------------+
2 rows in set (0.02 sec)
```

- **string manipulation**

```
mysql> select length ('sudham');
+-------------------+
| length ('sudham') |
+-------------------+
|                 6 |
+-------------------+
1 row in set (0.00 sec)
```

```
mysql> select lower("PRATHAMESH");
+----------------------+
| lower("PRATHAMESH") |
+----------------------+
| prathamesh           |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> select upper('monu');
+---------------+
| upper('monu') |
+---------------+
| MONU          |
+---------------+
1 row in set (0.00 sec)
```

```
mysql> select insert('sudham ',1,2,'prathamesh');
+-----------------------------------+
| insert('sudham ',1,2,'prathamesh') |
+-----------------------------------+
| prathameshdham                    |
+-----------------------------------+
1 row in set (0.01 sec)

mysql> mysql> select insert('monu',1,2,'babbu');
+---------------------------+
| insert('monu',1,2,'babbu') |
+---------------------------+
| babbunu                   |
+---------------------------+
1 row in set (0.00 sec)
```

```
mysql> select concat('pra','tham','esh');
+----------------------------+
| concat('pra','tham','esh') |
+----------------------------+
| prathamesh                 |
+----------------------------+
1 row in set (0.00 sec)

mysql> select concat('su','d','ham');
+------------------------+
| concat('su','d','ham') |
+------------------------+
| sudham                 |
+------------------------+
1 row in set (0.00 sec)
```

Conclusion - Executed various nested queries with aggregate functions, operators and with string manipulation.

Experiment No. 07

Aim - Write a program to implement Complex Queries.

Software Used - MySQL

Theory -

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc. A subquery is a SELECT statement nested in another statement.

See the following statement.

```
SELECT column_1 FROM table_1
WHERE column_1 = ( SELECT column_1 FROM table_2 );
```

You must use a pair of parentheses to enclose a subquery. Note that you can nest a subquery inside another subquery with a certain depth.

Typically, a subquery returns a single row as an atomic value, though it may return multiple rows for comparing values with the IN operator.

You can use a subquery in the SELECT, FROM, WHERE, and JOIN clauses.

Output (ScreenShot) -

Employee Table

```
mysql> select * from employee;
+------+------------+--------------+--------+------+
| eid  | emp_name   | street       | city   | cid  |
+------+------------+--------------+--------+------+
| 1001 | Prathamesh | Dwarli Road  | Kalyan |  108 |
| 1002 | Sudham     | Tisgaon Road | Kalyan |  151 |
| 1003 | Babbu      | Hajimalang   | Mumbai |  118 |
| 1004 | Monu       | Vijay Nagar  | Mumbai |  115 |
+------+------------+--------------+--------+------+
4 rows in set (0.02 sec)
```

Worker Table

```
mysql> select * from works;
+------+------+--------+
| eid  | cid  | salary |
+------+------+--------+
| 1001 |  108 |  30000 |
| 1002 |  151 |  32000 |
| 1003 |  118 |  31000 |
| 1004 |  115 |  28000 |
+------+------+--------+
4 rows in set (0.01 sec)
```

Company Table

```
mysql> select * from company;
+-----------+---------+-----+
| comp_name | city    | cid |
+-----------+---------+-----+
| Capgemini | Thane   | 108 |
| Google    | Mumbai  | 151 |
| Accenture | Airoli  | 118 |
| TCS       | Belapur | 115 |
+-----------+---------+-----+
4 rows in set (0.01 sec)
```

Manager Table

```
mysql> select * from manager;
+------+--------------+
| eid  | manager_name |
+------+--------------+
| 1001 | A            |
| 1002 | B            |
| 1003 | C            |
| 1004 | D            |
+------+--------------+
4 rows in set (0.01 sec)
```

- **Executed queries**

```
mysql> select manager_name from manager where eid between 1001 and 1003;
+--------------+
| manager_name |
+--------------+
| A            |
| B            |
| C            |
+--------------+
3 rows in set (0.00 sec)
```

```
mysql> select comp_name from company where cid between 100 and 120;
+------------+
| comp_name |
+------------+
| Capgemini |
| Accenture |
| TCS       |
+------------+
3 rows in set (0.00 sec)
```

```
mysql> select manager_name from manager where eid>1003;
+--------------+
| manager_name |
+--------------+
| D            |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> select salary from works where eid like 1001;
+--------+
| salary |
+--------+
|  30000 |
+--------+
1 row in set (0.00 sec)
```

Conclusion - Successfully Implemented Complex queries with the given databases.

Experiment No. 08

Aim - Write a program to implement and perform DCL and TCL commands.

Software Used - MySQL

Theory -

- **DCL commands:**

  Data Control Language(DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges.

  Privileges are of two types,

  System: This includes permissions for creating session, table, etc and all types of other system privileges.

  Object: This includes permissions for any command or query to perform any operation on the database tables.

  In DCL we have two commands,

  - GRANT: Used to provide any user access privileges or other privileges for the database.
  - REVOKE: Used to take back permissions from any user.

- **TCL commands:**

  Generally in SQLite transaction means it's a set of T-SQL statements that will execute together as a unit like a single T-SQL statement. If all these T-SQL statements are executed successfully without having any errors then the transaction will be committed and all the changes made by the transaction will be saved to the database permanently. In case, if any error occurred while executing these SQLite statements then the complete transaction will be rollbacked. Generally, SQLite is in an auto-commit mode that means SQLite automatically starts a transaction for each command, process, and commits the transaction changes automatically to the database. In case if we want to control

these transactions to maintain data consistency and to handle database errors then by using following commands we can disable auto-commit mode and explicitly start the transactions based on our requirements.

- START : It starts the transaction.
- COMMIT : It will commit the transaction that means all the changes saved to the Database.
- ROLLBACK : It will rollback the complete transaction

Output (ScreenShot) -

- **DCL**

```
mysql> CREATE USER 'sudham'@'localhost' IDENTIFIED BY 'Sudham_2412';
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW GRANTS FOR sudham@localhost;
+--------------------------------------------+
| Grants for sudham@localhost                |
+--------------------------------------------+
| GRANT USAGE ON *.* TO 'sudham'@'localhost' |
+--------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> GRANT ALL ON AIML51_SINGH.* TO sudham@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW GRANTS FOR sudham@localhost;
+------------------------------------------------------------+
| Grants for sudham@localhost                                |
+------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'sudham'@'localhost'                 |
| GRANT ALL PRIVILEGES ON `AIML51_SINGH`.* TO 'sudham'@'localhost' |
+------------------------------------------------------------+
2 rows in set (0.00 sec)
```

```
mysql> REVOKE SELECT, UPDATE ,INSERT ON AIML51_SINGH.* FROM sudham@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW GRANTS FOR sudham@localhost;
+--------------------------------------------------------------------------------
--------------------------------+
| Grants for sudham@localhost
                                |
+--------------------------------------------------------------------------------
--------------------------------+
| GRANT USAGE ON *.* TO 'sudham'@'localhost'
                                |
| GRANT DELETE, CREATE, DROP, REFERENCES, INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, EVENT, TRIGGER ON `AIML51
_SINGH`.* TO 'sudham'@'localhost' |
+--------------------------------------------------------------------------------
--------------------------------+
2 rows in set (0.00 sec)
```

- **TCL**

```
mysql>  CREATE TABLE t_school(ID INT, School_Name VARCHAR(40), Number_Of_Students INT, Number_Of_Teachers INT, Number_Of_Classrooms INT, Email
ID VARCHAR(40));
Query OK, 0 rows affected (0.28 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>  INSERT INTO t_school(ID, School_Name, Number_Of_Students, Number_Of_Teachers, Number_Of_Classrooms, EmailID) VALUES(1, "Boys Town Publ
ic School", 1000, 80, 12, "btps15@gmail.com"), (2, "Guru Govind Singh Public School", 800, 35, 15, "ggps25@gmail.com"), (3, "Delhi Public Scho
ol", 1200, 30, 10, "dps101@gmail.com"), (4, "Ashoka Universal School", 1110, 40, 40, "aus17@gmail.com"), (5, "Calibers English Medium School",
 9000, 31, 50, "cems@gmail.com");
Query OK, 5 rows affected (0.10 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT *FROM t_school;
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
| ID   | School_Name                    | Number_Of_Students | Number_Of_Teachers | Number_Of_Classrooms | EmailID          |
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
|    1 | Boys Town Public School        |              1000 |                80 |                  12 | btps15@gmail.com |
|    2 | Guru Govind Singh Public School |               800 |                35 |                  15 | ggps25@gmail.com |
|    3 | Delhi Public School            |              1200 |                30 |                  10 | dps101@gmail.com |
|    4 | Ashoka Universal School        |              1110 |                40 |                  40 | aus17@gmail.com  |
|    5 | Calibers English Medium School |              9000 |                31 |                  50 | cems@gmail.com   |
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
```

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT *FROM t_school;
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
| ID   | School_Name                    | Number_Of_Students | Number_Of_Teachers | Number_Of_Classrooms | EmailID          |
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
|    1 | Boys Town Public School        |              1000 |                80 |                  12 | btps15@gmail.com |
|    2 | Guru Govind Singh Public School |               800 |                35 |                  15 | ggps25@gmail.com |
|    3 | Delhi Public School            |              1200 |                30 |                  10 | dps101@gmail.com |
|    4 | Ashoka Universal School        |              1110 |                40 |                  40 | aus17@gmail.com  |
|    5 | Calibers English Medium School |              9000 |                31 |                  50 | cems@gmail.com   |
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
5 rows in set (0.00 sec)
```

```
mysql> SAVEPOINT savepoint_t_school;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from t_school;
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
| ID   | School_Name                    | Number_Of_Students | Number_Of_Teachers | Number_Of_Classrooms | EmailID          |
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
|    1 | Boys Town Public School        |              1000 |                80 |                  12 | btps15@gmail.com |
|    2 | Guru Govind Singh Public School |               800 |                35 |                  15 | ggps25@gmail.com |
|    3 | Delhi Public School            |              1200 |                30 |                  10 | dps101@gmail.com |
|    4 | Ashoka Universal School        |              1110 |                40 |                  40 | aus17@gmail.com  |
|    5 | Calibers English Medium School |              9000 |                31 |                  50 | cems@gmail.com   |
+------+--------------------------------+-------------------+-------------------+---------------------+------------------+
5 rows in set (0.00 sec)
```

```
mysql> insert into t_school values(6,'AA ENGLISH MEDIUM SCHOOL',1111,50,10,'PRATHAM@GMAIL.COM ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM t_school;
+------+-------------------------------+------------------+------------------+--------------------+--------------------+
| ID   | School_Name                   | Number_Of_Students | Number_Of_Teachers | Number_Of_Classrooms | EmailID            |
+------+-------------------------------+------------------+------------------+--------------------+--------------------+
|    1 | Boys Town Public School       |             1000 |               80 |                 12 | btps15@gmail.com   |
|    2 | Guru Govind Singh Public School |              800 |               35 |                 15 | ggps25@gmail.com   |
|    3 | Delhi Public School           |             1200 |               30 |                 10 | dps101@gmail.com   |
|    4 | Ashoka Universal School       |             1110 |               40 |                 40 | aus17@gmail.com    |
|    5 | Calibers English Medium School |             9050 |               31 |                 50 | cems@gmail.com     |
|    1 | Boys Town Public School       |             1000 |               80 |                 12 | btps15@gmail.com   |
|    2 | Guru Govind Singh Public School |              800 |               35 |                 15 | ggps25@gmail.com   |
|    3 | Delhi Public School           |             1200 |               30 |                 10 | dps101@gmail.com   |
|    4 | Ashoka Universal School       |             1110 |               40 |                 40 | aus17@gmail.com    |
|    5 | Calibers English Medium School |             9050 |               31 |                 50 | cems@gmail.com     |
|    6 | AA ENGLISH MEDIUM SCHOOL      |             1111 |               50 |                 10 | PRATHAM@GMAIL.COM  |
+------+-------------------------------+------------------+------------------+--------------------+--------------------+
11 rows in set (0.00 sec)

mysql> ROLLBACK TO t_school;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from t_school;
+------+-------------------------------+------------------+------------------+--------------------+--------------------+
| ID   | School_Name                   | Number_Of_Students | Number_Of_Teachers | Number_Of_Classrooms | EmailID            |
+------+-------------------------------+------------------+------------------+--------------------+--------------------+
|    1 | Boys Town Public School       |             1000 |               80 |                 12 | btps15@gmail.com   |
|    2 | Guru Govind Singh Public School |              800 |               35 |                 15 | ggps25@gmail.com   |
|    3 | Delhi Public School           |             1200 |               30 |                 10 | dps101@gmail.com   |
|    4 | Ashoka Universal School       |             1110 |               40 |                 40 | aus17@gmail.com    |
|    5 | Calibers English Medium School |             9050 |               31 |                 50 | cems@gmail.com     |
|    1 | Boys Town Public School       |             1000 |               80 |                 12 | btps15@gmail.com   |
|    2 | Guru Govind Singh Public School |              800 |               35 |                 15 | ggps25@gmail.com   |
|    3 | Delhi Public School           |             1200 |               30 |                 10 | dps101@gmail.com   |
|    4 | Ashoka Universal School       |             1110 |               40 |                 40 | aus17@gmail.com    |
|    5 | Calibers English Medium School |             9050 |               31 |                 50 | cems@gmail.com     |
+------+-------------------------------+------------------+------------------+--------------------+--------------------+
10 rows in set (0.00 sec)
```

Conclusion - Successfully implemented and performed DCL and TCL commands.

Experiment No. 09

Aim - Write a program to implement Procedure and Functions.

Software Used - MySQL

Theory -

- **MySQL Stored Procedure**

  A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database. It is a subroutine or a subprogram in the regular computing language. A procedure always contains a name, parameter lists, and SQL statements. We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc. It was first introduced in MySQL version 5. Presently, it can be supported by almost all relational database systems.

  If we consider the enterprise application, we always need to perform specific tasks such as database cleanup, processing payroll, and many more on the database regularly. Such tasks involve multiple SQL statements for executing each task. This process might be easy if we group these tasks into a single task. We can fulfill this requirement in MySQL by creating a stored procedure in our database.

  A procedure is called a recursive stored procedure when it calls itself. Most database systems support recursive stored procedures. But, it is not supported well in MySQL.

  Features:

  - Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.

  - Stored procedure reduces the traffic between application and database server. Because the application has to send only the stored

procedure's name and parameters instead of sending multiple SQL statements. Stored procedures are reusable and transparent to any applications.

- A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permissions on the database tables.

Output (ScreenShot) -

```
mysql> delimiter @@
mysql> select * from students;
    -> @@
+---------+------------+-------+--------+
| roll_no | name       | marks | status |
+---------+------------+-------+--------+
|      51 | sudham     |    80 | pass   |
|       8 | prathamesh |    85 | pass   |
|     151 | monu       |    85 |        |
|     108 | babbu      |    80 |        |
+---------+------------+-------+--------+
4 rows in set (0.00 sec)

mysql> create procedure updatestatus1 (in mark1 int)
    -> begin
    -> declare mymark int;
    -> set mymark=mark1;
    -> if mymark>35 then
    -> update students set status='pass' where marks=mymark;
    -> else
    -> update students set status = 'fail' where marks=mymark;
    -> end if;
    -> end;
    -> @@
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> call updatestatus1(85);
    -> @@
Query OK, 1 row affected (0.00 sec)

mysql> select * from students;
    -> @@
+---------+------------+-------+--------+
| roll_no | name       | marks | status |
+---------+------------+-------+--------+
|      51 | sudham     |    80 | pass   |
|       8 | prathamesh |    85 | pass   |
|     151 | monu       |    85 | pass   |
|     108 | babbu      |    80 |        |
+---------+------------+-------+--------+
4 rows in set (0.00 sec)

mysql> call updatestatus1(80);
    -> @@
Query OK, 1 row affected (0.00 sec)

mysql> select * from students;
    -> @@
+---------+------------+-------+--------+
| roll_no | name       | marks | status |
+---------+------------+-------+--------+
|      51 | sudham     |    80 | pass   |
|       8 | prathamesh |    85 | pass   |
|     151 | monu       |    85 | pass   |
|     108 | babbu      |    80 | pass   |
+---------+------------+-------+--------+
4 rows in set (0.00 sec)
```

Conclusion - Successfully implement Procedure and Functions.

Experiment No. 10

Aim - Write a program to implement Views and Triggers.

Software Used - MySQL

Theory -

- **Triggers**

  An SQLite trigger is a named database object that is executed automatically when an INSERT, UPDATE or DELETE statement is issued against the associated table. You often use triggers to enable sophisticated auditing. For example, you want to log the changes in the sensitive data such as salary and address whenever it changes. In addition, you use triggers to enforce complex business rules centrally at the database level and prevent invalid transactions.

  CREATE TRIGGER [IF NOT EXISTS] trigger_name [BEFORE|AFTER|INSTEAD OF] [INSERT|UPDATE|DELETE] ON table_name [WHEN condition] BEGIN statements; END; If you combine the time when the trigger is fired and the event that causes the trigger to be fired, you have a total of 9 possibilities:

  - BEFORE INSERT
  - AFTER INSERT
  - BEFORE UPDATE
  - AFTER UPDATE
  - BEFORE DELETE
  - AFTER DELETE
  - INSTEAD OF INSERT
  - INSTEAD OF DELETE
  - INSTEAD OF UPDATE

- **Views**

  In database theory, a view is a result set of a stored query. A view is the way to pack a query into a named object stored in the database. You can access the data of the underlying tables through a view. The tables that the query in the view definition refers to are called base tables. A view is useful in some cases: - First, views provide an abstraction layer over tables. You can add and remove the columns in the view without touching the schema of the underlying tables. - Second, you can use views to encapsulate complex queries with joins to simplify the data access. SQLite view is read only. It means you cannot use INSERT, DELETE, and UPDATE statements to update data in the base tables through the view.

  Syntax:

  CREATE [TEMP] VIEW [IF NOT EXISTS] view_name[(column-name-list)] AS select-statement;

Output (ScreenShot) -

- **Triggers**

```
mysql> CREATE TABLE employee(
    ->     name varchar(45) NOT NULL,
    ->     occupation varchar(35) NOT NULL,
    ->     working_date date,
    ->     working_hours varchar(10)
    -> );
Query OK, 0 rows affected (0.31 sec)
```

```
mysql> insert into table values(name,occupation,working_date,working_hours)(
mysql> INSERT INTO employee VALUES
    -> ('Robin', 'Scientist', '2020-10-04', 12),
    -> ('Warner', 'Engineer', '2020-10-04', 10),
    -> ('Peter', 'Actor', '2020-10-04', 13),
    -> ('Marco', 'Doctor', '2020-10-04', 14),
    -> ('Brayden', 'Teacher', '2020-10-04', 12),
    -> ('Antonio', 'Business', '2020-10-04', 11);
Query OK, 6 rows affected (0.16 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

```
mysql> delimiter //
mysql> Create Trigger before_insert_empworkinghours
    -> BEFORE INSERT ON employee FOR EACH ROW
    -> BEGIN
    -> IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
    -> END IF;
    -> END //
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO employee VALUES
    -> ('Markus', 'Former', '2020-10-08', 14);
    ->
    -> ;
    -> ;
    -> //
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO employee VALUES     ('Markus', 'Former', '2020-10-08', -12)//
Query OK, 1 row affected (0.04 sec)
```

```
mysql> select * from employee
    -> //
+---------+------------+--------------+---------------+
| name    | occupation | working_date | working_hours |
+---------+------------+--------------+---------------+
| Robin   | Scientist  | 2020-10-04   | 12            |
| Warner  | Engineer   | 2020-10-04   | 10            |
| Peter   | Actor      | 2020-10-04   | 13            |
| Marco   | Doctor     | 2020-10-04   | 14            |
| Brayden | Teacher    | 2020-10-04   | 12            |
| Antonio | Business   | 2020-10-04   | 11            |
| Markus  | Former     | 2020-10-08   | 14            |
| Markus  | Former     | 2020-10-08   | 0             |
+---------+------------+--------------+---------------+
8 rows in set (0.02 sec)
```

- **Views**

```
mysql> create view v32 as select name,working_hours from employee;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from v32;
+---------+---------------+
| name    | working_hours |
+---------+---------------+
| Robin   | 12            |
| Warner  | 10            |
| Peter   | 13            |
| Marco   | 14            |
| Brayden | 12            |
| Antonio | 11            |
| Markus  | 14            |
| Markus  | 0             |
+---------+---------------+
8 rows in set (0.00 sec)
```

Conclusion - Successfully implement Views and Triggers.

Query No. 01

>Write SQL queries for the given database.

-Employee(eid, emp-name, street, city)

-Works(eid, cid, salary)

-Company(cid, comp-name, city)

-Manager(eid, manager-name)

```
mysql> select * from employee;
+------+------------+-------------+--------+------+
| eid  | emp_name   | street      | city   | cid  |
+------+------------+-------------+--------+------+
| 1001 | Prathamesh | Dwarli Road | Kalyan |  108 |
| 1002 | Sudham     | Tisgaon Road | Kalyan |  151 |
| 1003 | Babbu      | Hajimalang  | Mumbai |  118 |
| 1004 | Monu       | Vijay Nagar | Mumbai |  115 |
+------+------------+-------------+--------+------+
4 rows in set (0.00 sec)

mysql> select * from works;
+------+------+--------+
| eid  | cid  | salary |
+------+------+--------+
| 1001 |  108 |  30000 |
| 1002 |  151 |  32000 |
| 1003 |  118 |  31000 |
| 1004 |  115 |  28000 |
+------+------+--------+
4 rows in set (0.00 sec)

mysql> select * from company;
+-----------+---------+-----+
| comp_name | city    | cid |
+-----------+---------+-----+
| Capgemini | Thane   | 108 |
| Google    | Mumbai  | 151 |
| Accenture | Airoli  | 118 |
| TCS       | Belapur | 115 |
+-----------+---------+-----+
4 rows in set (0.00 sec)

mysql> select * from manager;
+------+--------------+
| eid  | manager_name |
+------+--------------+
| 1001 | A            |
| 1002 | B            |
| 1003 | C            |
| 1004 | D            |
+------+--------------+
4 rows in set (0.00 sec)
```

(i) Find the names of all the employees having 'S' as the first letter in their names.

```
mysql> #FIRST QUERY
mysql> select emp_name from employee
    -> where emp_name like 's%';
+-----------+
| emp_name |
+-----------+
| Sudham   |
+-----------+
1 row in set (0.00 sec)

mysql> #FIRST QUERY
mysql> select emp_name from employee
    -> where emp_name like 'p%';
+------------+
| emp_name   |
+------------+
| Prathamesh |
+------------+
1 row in set (0.00 sec)
```

(ii) Display the annual salary of all the employees.

```
mysql> #SECOND QUERY
mysql> select salary*12 from works;
+-----------+
| salary*12 |
+-----------+
|    360000 |
|    384000 |
|    372000 |
|    336000 |
+-----------+
4 rows in set (0.00 sec)
```

(iii) Find the name, street and city of all employees who work for "Accenture" and earn more than 30,000.

```
mysql> #THIRD QUERY
mysql> select emp_name,street,city from employee e,works w
    -> where e.cid = w.cid and w.salary >30000 and w.cid in
    -> (select cid from company where comp_name = 'Accenture');
+----------+------------+--------+
| emp_name | street     | city   |
+----------+------------+--------+
| Babbu    | Hajimalang | Mumbai |
+----------+------------+--------+
1 row in set (0.01 sec)
```

(iv) Give the total number of employees.

```
mysql> #FOURTH QUERY
mysql> select count(*) from employee;
+----------+
| count(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

Query No. 02

STUDENT(ROLLNO,NAME,FATHER_NAME,BRANCH)

BOOK(ISBN,AUTHOR,TITLE,AUTHOR,PUBLISHER)

ISSUE(ROLLNO,ISBN,DATE_OF_ISSUE)

```
mysql> select *from student;
+---------+------------+-------------+--------+------+
| roll_no | name       | father_name | branch | isbn |
+---------+------------+-------------+--------+------+
|      51 | Sudham     | Dharmendra  | AIML   |  101 |
|       8 | Prathamesh | Shivaji     | IOT    |  102 |
|     151 | Monu       | Dharmesh    | DS     |  103 |
|     118 | Babbu      | Shiva       | IT     |  104 |
+---------+------------+-------------+--------+------+
4 rows in set (0.00 sec)

mysql> select *from book;
+------+-----------+----------------+-----------+
| isbn | author    | title          | publisher |
+------+-----------+----------------+-----------+
|  101 | SINGH     | FIRST EDITION  | ABC       |
|  102 | CHIKANKAR | SECOND EDITION | EFG       |
|  103 | PATEL     | THIRD EDITION  | XYZ       |
|  104 | SARDAR    | LAST EDITION   | MNO       |
+------+-----------+----------------+-----------+
4 rows in set (0.00 sec)

mysql> select *from issue;
+---------+------+---------------+
| roll_no | isbn | Date_of_issue |
+---------+------+---------------+
|      51 |  101 | 2018-12-15    |
|       8 |  102 | 2019-06-23    |
|     151 |  103 | 2017-06-12    |
|     118 |  104 | 2020-04-12    |
+---------+------+---------------+
4 rows in set (0.00 sec)
```

a)list roll number and name of all students of branch IT

```
mysql> #FIRST QUERY
mysql> select roll_no,name
    -> from student
    -> where branch='IT';
+---------+-------+
| roll_no | name  |
+---------+-------+
|     118 | Babbu |
+---------+-------+
1 row in set (0.00 sec)
```

b) find name of students issued books in name of publisher "XYZ"

```
mysql> #SECOND QUERY
mysql> select name
    -> from student s,book b
    -> where b.publisher='XYZ'
    -> and s.isbn=b.isbn;
+------+
| name |
+------+
| Monu |
+------+
1 row in set (0.00 sec)
```

c)list title of all books issued on or before "31-dec-2019"

```
mysql> #THIRD QUERY
mysql> SELECT title
    -> from book b, issue i
    -> where Date_of_issue < '2019-12-31'
    -> and b.isbn=i.isbn;
+----------------+
| title          |
+----------------+
| FIRST EDITION  |
| SECOND EDITION |
| THIRD EDITION  |
+----------------+
3 rows in set (0.01 sec)
```

d)list title of all books and their author issued by student "alice"

Command Prompt - mysql -u root -p

```
mysql> #FOURTH QUERY
mysql> select title ,author
    -> from book b, student s
    -> where s.name ='Sudham'
    -> and b.isbn=s.isbn;
+---------------+--------+
| title         | author |
+---------------+--------+
| FIRST EDITION | SINGH  |
+---------------+--------+
1 row in set (0.00 sec)

mysql> select title ,author
    -> from book b, student s
    -> where s.name ='Prathamesh'
    -> and b.isbn=s.isbn;
+----------------+-----------+
| title          | author    |
+----------------+-----------+
| SECOND EDITION | CHIKANKAR |
+----------------+-----------+
1 row in set (0.00 sec)
```