INDEX

S.NO.	DATE	Exercises	Page No.	Marks	Sign
1		DDL Commands – CREATE, ALTER, DROP			
2		DDL Commands with Constraints – PRIMARY, FOREIGN KEY, UNIQUE, CHECK			
3		DML Commands – INSERT, SELECT, UPDATE, DELETE			
4		SELECT with various clause – WHERE, pattern matching			
5		SELECT with various clause – BETWEEN, IN, Aggregate Function			
6		SELECT with various clause – GROUP BY, HAVING, ORDER BY			
7		SubQuery& Correlated Query			
8		Joins – EquiJoin, InnerJoin, OuterJoin			
9		VIEW, INDEX, SEQUENCE			
10		Simple programming exercises using CASE, IF, ITERATE, LEAVE, LOOP			
11		Simple programming exercises using REPEAT, WHILE			
12		TCL Commands – COMMIT, ROLLBACK, SAVEPOINT			
13		DCL Commands – GRANT, REVOKE			
14		Procedures			
15		Function			
16		Cursors			
17		Triggers			
18		Database Connectivity – Using PhP& MySQL			

Ex.No. 1
Date:

DDL Commands – CREATE, ALTER, DROP

Aim:

To Create, Alter and Drop the table using Data Definition Language.

Description:

Data Definition Language (DDL) statements are used to define the database structure or schema.
DDL Commands: Create, Alter, Drop, Rename, Truncate

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- RENAME - rename an object

SYNTAX:

CREATE TABLE

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
);
```

ALTER A TABLE

To add a column in a table
ALTER TABLE table_name
ADD column_namedatatype;

To delete a column in a table
ALTER TABLE table_name
DROP COLUMN column_name;

DROP TABLE

DROP TABLE table_name;

TRUNCATE TABLE

TRUNCATE TABLE table_name;

Questions:

1) Create a table name STUDENT with following structure.

#	Column Name	Description	Data Type
1	RegNo	Registration Number	NUMBER(3)
2	Name	Student Name	VARCHAR(15)
3	Gender	Gender of the student	CHAR(1)
4	DOB	Date of Birth	DATE
5	MobileNo	Mobile Number	NUMBER(10)
6	City	Location of stay	VARCHAR(15)

2) Create a table name FACULTY with following structure.

#	Column Name	Description	Data Type
1	FacNo	Faculty Identifier	VARCHAR(4)
2	FacName	Faculty Name	VARCHAR(15)
3	Gender	Gender of faculty	CHAR(1)
4	DOB	Date of Birth	DATE
5	DOJ	Date of Join	DATE
6	MobileNo	Mobile Number	NUMBER(10)

3) Create a table name DEPARTMENT with following structure.

#	Column Name	Description	Data Type
1	DeptNo	Department Identifier	VARCHAR(4)
2	DeptName	Department Name	VARCHAR(15)
3	DeptHead	Department Head	VARCHAR(4)

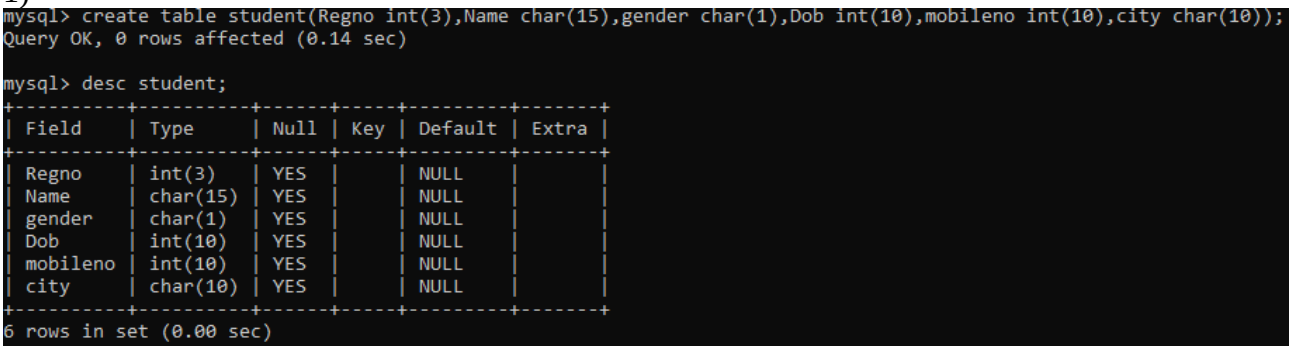
4) Create a table name COURSE with following structure.

#	Column Name	Description	Data Type
1	CourseNo	Course Identifier	VARCHAR(3)
2	CourseDesc	Course Description	VARCHAR(14)
3	CourseType	Course Type	CHAR(1)
4	SemNo	Semester Number	CHAR(1)
5	HallNo	Hall Number	VARCHAR(4)
6	FacNo	Faculty Identifier	VARCHAR(4)

5) Modify the table FACULTY by adding a column name DeptNo of datatype VARCHAR(4)

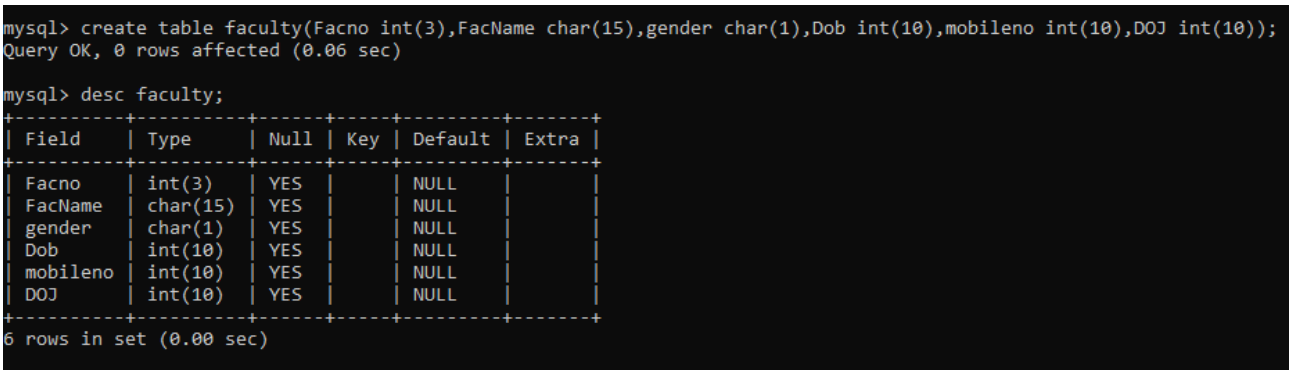
OUTPUTS:

1)



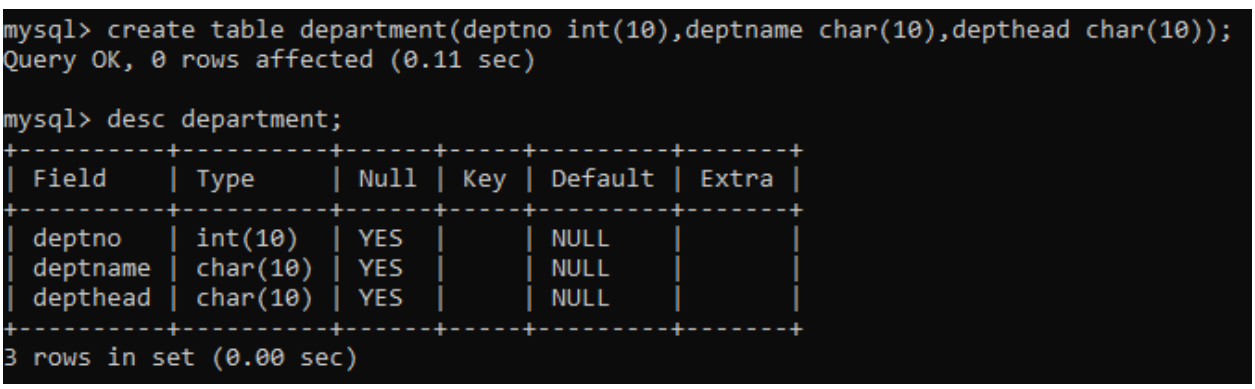
```
mysql> create table student(Regno int(3),Name char(15),gender char(1),Dob int(10),mobileno int(10),city char(10));
Query OK, 0 rows affected (0.14 sec)

mysql> desc student;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| Regno    | int(3)   | YES  |     | NULL    |       |
| Name     | char(15) | YES  |     | NULL    |       |
| gender   | char(1)  | YES  |     | NULL    |       |
| Dob      | int(10)  | YES  |     | NULL    |       |
| mobileno | int(10)  | YES  |     | NULL    |       |
| city     | char(10) | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

2)



```
mysql> create table faculty(Facno int(3),FacName char(15),gender char(1),Dob int(10),mobileno int(10),DOJ int(10));
Query OK, 0 rows affected (0.06 sec)

mysql> desc faculty;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| Facno    | int(3)   | YES  |     | NULL    |       |
| FacName  | char(15) | YES  |     | NULL    |       |
| gender   | char(1)  | YES  |     | NULL    |       |
| Dob      | int(10)  | YES  |     | NULL    |       |
| mobileno | int(10)  | YES  |     | NULL    |       |
| DOJ      | int(10)  | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

3)



```
mysql> create table department(deptno int(10),deptname char(10),depthead char(10));
Query OK, 0 rows affected (0.11 sec)

mysql> desc department;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| deptno   | int(10)  | YES  |     | NULL    |       |
| deptname | char(10) | YES  |     | NULL    |       |
| depthead | char(10) | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

4)

```
mysql> create table course(courseno int(3),coursedesc char(15),coursetype char(1),semno int(10),hallno int(10),Facno int(10));
Query OK, 0 rows affected (0.09 sec)

mysql> desc course;
+------------+---------+------+-----+---------+-------+
| Field      | Type    | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
| courseno   | int(3)  | YES  |     | NULL    |       |
| coursedesc | char(15)| YES  |     | NULL    |       |
| coursetype | char(1) | YES  |     | NULL    |       |
| semno      | int(10) | YES  |     | NULL    |       |
| hallno     | int(10) | YES  |     | NULL    |       |
| Facno      | int(10) | YES  |     | NULL    |       |
+------------+---------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

5)

```
mysql> alter table faculty add dept char(10);
Query OK, 2 rows affected (0.09 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> desc faculty;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| Facno    | int(3)   | NO   | PRI | NULL    |       |
| FacName  | char(15) | YES  |     | NULL    |       |
| gender   | char(1)  | YES  |     | NULL    |       |
| Dob      | int(10)  | YES  |     | NULL    |       |
| mobileno | int(10)  | YES  |     | NULL    |       |
| DOJ      | int(10)  | YES  |     | NULL    |       |
| dept     | char(10) | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

RESULT:

Tables are created, altered and modified using DDL commands.

Ex.No. 2
Date:

DDL Commands with Constraints – PRIMARY, FOREIGN KEY, UNIQUE, CHECK

AIM:

To add the constraints like primary key, foreign key, unique key and check using DDL commands.

Description:

PRIMARY KEY:

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only one primary key, which may consist of single or multiple fields.

FOREIGN KEY:

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

UNIQUE Constraint:

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

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 single column it allows 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.

PRIMARY:

```
ALTER TABLE table_name
ADD PRIMARY KEY(primary_key_column);
```

FOREIGN KEY:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
FOREIGN KEY foreign_key_name (columns)
REFERENCES parent_table(columns)
ON DELETE action
ON UPDATE action
```

UNIQUE:

```
CREATE TABLE table_1(
...
column_name_1 data_type,
...
UNIQUE(column_name_1)
);
```

CHECK

```
CREATE TABLE IF NOT EXISTS parts (
part_no VARCHAR(18) PRIMARY KEY,
description VARCHAR(40),
cost DECIMAL(10 , 2 ) NOT NULL CHECK(cost > 0), price DECIMAL (10,2) NOT
NULL
);
```

Questions:

1) Alter the table STUDENT with following structure.

#	Column Name	Constraints
1	RegNo	PRIMARY KEY
2	MobileNo	NOT NULL

2) Alter the table name FACULTY with following structure. The DeptNo in this table refers the DeptNo in the DEPARTMENT table.

#	Column Name	Constraints
1	FacNo	PRIMARY KEY
2	Gender	CHECK 'M' or 'F'

3)After the FACULTY table is successfully created, test if you can add a constraint FOREIGN KEY to the DeptNo of this table.

4)Alter the table name DEPARTMENT with following structure.

#	Column Name	Constraint
1	DeptNo	PRIMARY KEY

5) Alter the table name COURSE with following structure.

#	Column Name	Constraint
1	CourseNo	PRIMARY KEY
2	SemNo	1 to 6

OUTPUTS:

1)

```
mysql> alter table student add primary key(city);
Query OK, 0 rows affected (0.14 sec)
mysql> desc student;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| Regno    | int(3)   | YES  |     | NULL    |       |
| Name     | char(15) | YES  |     | NULL    |       |
| gender   | char(1)  | YES  |     | NULL    |       |
| Dob      | int(10)  | YES  |     | NULL    |       |
| mobileno | int(10)  | YES  |     | NULL    |       |
| city     | char(10) | NO   | PRI | NULL    |       |
+----------+----------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

2)

```
mysql> alter table faculty add primary key(facno);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table faculty add check(gender='M'or'F');
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc faculty;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| Facno    | int(3)   | NO   | PRI | NULL    |       |
| FacName  | char(15) | YES  |     | NULL    |       |
| gender   | char(1)  | YES  |     | NULL    |       |
| Dob      | int(10)  | YES  |     | NULL    |       |
| mobileno | int(10)  | YES  |     | NULL    |       |
| DOJ      | int(10)  | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

4)

```
mysql> alter table department add primary key(deptno);
Query OK, 0 rows affected (0.11 sec)
mysql> desc department;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| deptno   | int(10)  | NO   | PRI | NULL    |       |
| deptname | char(10) | YES  |     | NULL    |       |
| depthead | char(10) | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

5)

```
mysql> alter table course add primary key(courseno);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table course add check(semno>=1&&semno<=6);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc course;
+------------+----------+------+-----+---------+-------+
| Field      | Type     | Null | Key | Default | Extra |
+------------+----------+------+-----+---------+-------+
| courseno   | int(3)   | NO   | PRI | NULL    |       |
| coursedesc | char(15) | YES  |     | NULL    |       |
| coursetype | char(1)  | YES  |     | NULL    |       |
| semno      | int(10)  | YES  |     | NULL    |       |
| hallno     | int(10)  | YES  |     | NULL    |       |
| Facno      | int(10)  | YES  |     | NULL    |       |
+------------+----------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Result:

DDL Commands with Primary, Foreign, Unique, Check constraints are updated and verified.

Ex.No. : 3
Date:

DML Commands – INSERT, SELECT, UPDATE, DELETE

Aim:

To perform Data Manipulation Language (DML) Commands such as INSERT, SELECT, UPDATE, DELETE in the table.

Description:

Data Manipulation Language (DML) statements are used for managing data within schema objects. DML Commands: Insert , Update, Delete, Select

- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- SELECT - retrieve data from the a database

INSERT:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...);
( or )
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...);
```

UPDATE:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value;
```

DELETE:

```
DELETE FROM table_name
WHERE some_column=some_value;
```

SELECT:

```
SELECT column_name(s)
FROM table_name;
```

Questions:

1. Populate all the five tables with your own data.
2. Update the value of student name whose register number is ‘191711342’
3. Delete the record in the table FACULTY, who resigned her job.
4. Modify the date of birth for the faculty whose name is 'RAM' with a value ‘1983-05-01’.
5. Remove all faculty who are having over 65 years
6. View all the records from the five tables.

OUTPUTS:

1)

```
mysql> SELECT * FROM college;
+------+---------+-----------+------+------+------+-------+
| sno  | name    | regno     | dept | age  | DOB  | facno |
+------+---------+-----------+------+------+------+-------+
|    1 | YUGA    | 191811316 | cse  |   19 | 2000 |   200 |
|    2 | SRI     | 191811382 | cse  |   19 | 2000 |   201 |
|    3 | ABDUL   | 191811362 | cse  |   20 | 1999 |   202 |
|    4 | SUPRIYA | 191811322 | cse  |   20 | 1999 |   203 |
+------+---------+-----------+------+------+------+-------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM facl;
+-------+----------+--------+------+-----------+------+
| FacNo | FactName | Gender | DOB  | MobileNo  | DOJ  |
+-------+----------+--------+------+-----------+------+
|     1 | YUGA     | M      | 2000 | 123456789 | 2001 |
+-------+----------+--------+------+-----------+------+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM department;
+---------+-----------+-----------+
| deptno  | deptname  | depthead  |
+---------+-----------+-----------+
|       1 | cse       | hari      |
+---------+-----------+-----------+
1 row in set (0.00 sec)
```

```
mysql> insert into course values(1,'prgm','u',2000,201,2001);
Query OK, 1 row affected (0.00 sec)

mysql> select * from course;
+----------+------------+------------+-------+--------+-------+
| courseno | coursedesc | coursetype | semno | hallno | Facno |
+----------+------------+------------+-------+--------+-------+
|        1 | prgm       | u          |  2000 |    201 |  2001 |
+----------+------------+------------+-------+--------+-------+
1 row in set (0.00 sec)
```

2)

```
mysql> update student set regno=324 where name='mukesh';
Query OK, 0 rows affected (0.03 sec)
```

3)

```
mysql> delete from faculty where FacName='Ramu';
Query OK, 1 row affected (0.00 sec)
```

4)

```
mysql> select * from faculty;
+-------+---------+--------+------+----------+------+
| Facno | FacName | gender | Dob  | mobileno | DOJ  |
+-------+---------+--------+------+----------+------+
|     1 | ramu    | M      | 2000 | 90515252 | 2001 |
+-------+---------+--------+------+----------+------+
1 row in set (0.00 sec)
```

5)

```
mysql> delete from faculty where Facno='2';
Query OK, 1 row affected (0.06 sec)

mysql> select * from faculty;
+-------+---------+--------+------+----------+------+
| Facno | FacName | gender | Dob  | mobileno | DOJ  |
+-------+---------+--------+------+----------+------+
|     1 | ramu    | M      | 2000 | 90515252 | 2001 |
+-------+---------+--------+------+----------+------+
1 row in set (0.00 sec)
```

6)

```
mysql> select * from student;
+------+-------------+-------------+
| sno  | column_name | constraints |
+------+-------------+-------------+
|    1 | RegNo       |             |
|    1 | RegNo       | xyz         |
+------+-------------+-------------+
2 rows in set (0.03 sec)
```

```
mysql> select * from faculty;
+-------+---------+--------+------+----------+------+
| Facno | FacName | gender | Dob  | mobileno | DOJ  |
+-------+---------+--------+------+----------+------+
|     1 | ramu    | M      | 2000 | 90515252 | 2001 |
+-------+---------+--------+------+----------+------+
1 row in set (0.00 sec)

mysql> select * from course;
+----------+------------+------------+-------+--------+-------+
| courseno | coursedesc | coursetype | semno | hallno | Facno |
+----------+------------+------------+-------+--------+-------+
|        1 | prgm       | u          |  2000 |    201 |  2001 |
+----------+------------+------------+-------+--------+-------+
1 row in set (0.00 sec)

mysql> select * from department;
+--------+----------+----------+
| deptno | deptname | depthead |
+--------+----------+----------+
|      1 | cse      | vasu     |
+--------+----------+----------+
1 row in set (0.00 sec)
```

RESULT:

Data Manipulation Language (DML) Commands such as INSERT, SELECT, UPDATE, DELETE are performed in the five tables.

Ex. No.: 4
Date:

SELECT with various clause – WHERE, pattern matching

AIM:

To view the records from the tables using SELECT commands with WHERE Clause and Pattern matching.

DESCRIPTION:

The SELECT statement allows you to get the data from tables. A table consists of rows and columns like a spreadsheet. Often, you want to see a subset rows, a subset of columns, or a combination of two. The result of the SELECT statement is called a result set that is a list of rows, each consisting of the same number of columns.

SELECT:

```
SELECT
column_1, column_2, ...
FROM
table_1
[INNER | LEFT |RIGHT] JOIN table_2 ON conditions
WHERE
conditions
GROUP BY column_1
HAVING group_conditions
ORDER BY column_1
LIMIT offset, length;
```

The `SELECT` statement consists of several clauses as explained in the following list:

- SELECT followed by a list of comma-separated columns or an asterisk (*) to indicate that you want to return all columns.
- FROM specifies the table or view where you want to query the data.
  - JOIN gets related data from other tables based on specific join conditions.
  - WHERE clause filters row in the result set.
- GROUP BY clause groups a set of rows into groups and applies aggregate functions on each group.
  - HAVING clause filters group based on groups defined by GROUP BY clause.
- ORDER BY clause specifies a list of columns for sorting.
- LIMIT constrains the number of returned rows.

Questions:

WHERE:

1. The student counsellor wanted to display the registration number, student name and date of birth for all the students.
2. The controller of examinations wanted to list all the female students
3. Who are the boy students registered for course with the course number “C001“
4. Display all faculty details joined before “November 2014”
5. Display all the courses not allotted to halls

LIKE:

6. List the students whose name ends with the substring “ma”
7. Display all students whose name contains the substring “ma”
8. Find all the students who are located in cities having “Sal” as substring
9. Display the students whose names do not contain six letters.
10. Find all the students whose names contains “th”

OUTPUTS:

1)

```
mysql> SELECT  * FROM student;
+------+-------------------+------+------+------+-------+--------+
| sno  | name              | dept | age  | DOB  | Facno | gender |
+------+-------------------+------+------+------+-------+--------+
|    1 | DHONI             | cse  |   19 | 2000 |     7 | M      |
|    2 | virat             | cse  |   19 | 2000 |    71 | M      |
|    3 | sindhu            | cse  |   18 | 2001 |    71 | F      |
|    4 | VIJAY_DEVARAKONDA | cse  |   21 | 1998 |     3 | M      |
+------+-------------------+------+------+------+-------+--------+
4 rows in set (0.00 sec)
```

2)

```
mysql> select name from student where gender='f';
+------+
| name |
+------+
| nani |
| ram  |
+------+
```

3)

```
mysql> select * from course;
+----------+------------+------------+-------+--------+-------+
| courseno | coursedesc | coursetype | semno | hallno | Facno |
+----------+------------+------------+-------+--------+-------+
|        1 | prgm       | u          |  2000 |    201 |  2001 |
+----------+------------+------------+-------+--------+-------+
1 row in set (0.00 sec)
```

4)

```
mysql> select * from course;
+----------+------------+------------+-------+--------+-------+
| courseno | coursedesc | coursetype | semno | hallno | Facno |
+----------+------------+------------+-------+--------+-------+
|        1 | prgm       | u          |  2000 |    201 |  2001 |
+----------+------------+------------+-------+--------+-------+
1 row in set (0.00 sec)
```

5)

```
mysql> select * from course;
+----------+------------+------------+-------+--------+-------+
| courseno | coursedesc | coursetype | semno | hallno | Facno |
+----------+------------+------------+-------+--------+-------+
|        1 | prgm       | u          |  2000 |    201 |  2001 |
+----------+------------+------------+-------+--------+-------+
1 row in set (0.00 sec)
```

6)

```
mysql> select * from student where name like'%sh';
```

7)

```
mysql> select * from student where name like'ma%';
```

LIKE:

The LIKE operator is commonly used to select data based on patterns. Using the LIKE operator in the right way is essential to increase the query performance.

The LIKE operator allows you to select data from a table based on a specified pattern. Therefore, the LIKE operator is often used in the WHERE clause of the SELECT statement.

MySQL provides two wildcard characters for using with the LIKE operator, the percentage % and underscore _ .

- The percentage (%) wildcard allows you to match any string of zero or more characters.
- The underscore (_) wildcard allows you to match any single character.

8)

```
mysql> SELECT  * FROM student;
+------+--------------------+------+------+------+-------+--------+
| sno  | name               | dept | age  | DOB  | Facno | gender |
+------+--------------------+------+------+------+-------+--------+
|    1 | DHONI              | cse  |   19 | 2000 |     7 | M      |
|    2 | virat              | cse  |   19 | 2000 |    71 | M      |
|    3 | sindhu             | cse  |   18 | 2001 |    71 | F      |
|    4 | VIJAY_DEVARAKONDA  | cse  |   21 | 1998 |     3 | M      |
+------+--------------------+------+------+------+-------+--------+
4 rows in set (0.00 sec)
```

9)

```
mysql> SELECT  * FROM student;
+------+--------------------+------+------+------+-------+--------+
| sno  | name               | dept | age  | DOB  | Facno | gender |
+------+--------------------+------+------+------+-------+--------+
|    1 | DHONI              | cse  |   19 | 2000 |     7 | M      |
|    2 | virat              | cse  |   19 | 2000 |    71 | M      |
|    3 | sindhu             | cse  |   18 | 2001 |    71 | F      |
|    4 | VIJAY_DEVARAKONDA  | cse  |   21 | 1998 |     3 | M      |
+------+--------------------+------+------+------+-------+--------+
4 rows in set (0.00 sec)
```

10)

```
mysql> select * from student where 'name' like '%h';
Empty set (0.00 sec)
```

RESULT:

The records from the tables are displayed using SELECT commands with WHERE Clause and Pattern matching.

Ex. No. : 5
Date:

SELECT with various clause – BETWEEN, IN, Aggregate function

AIM:

To view the records from the tables using SELECT commands with BETWEEN, IN, Aggregate functions.

DESCRIPTION:

The BETWEEN operator allows you to specify a range to test. We often use the BETWEEN operator in the WHERE clause of the SELECT, INSERT, UPDATE, and DELETE statements.

The IN operator allows you to determine if a specified value matches any one of a list or a sub query.

MySQL provides many aggregate functions that include AVG, COUNT, SUM, MIN, MAX, etc. An aggregate function ignores NULL values when it performs calculation except for the COUNT function.

BETWEEN operator:

```
SELECT
column1,column2,...
FROM
table_name
WHERE expr [NOT] BETWEEN begin_expr AND end_expr;
```

The *expr* is the expression to test in the range that is defined by *begin_expr* and *end_expr*.

`IN` operator:

```
SELECT
column1,column2,...
FROM
table_name
WHERE (expr|column_1) IN ('value1','value2',...);
```

Questions:

IN & BETWEEN

1. List the type of the courses “Statistics” and “Programming”
2. The instructor wants to know the CourseNos whose scores are in the range 50 to 80

AGGREGATE

1. Find the average mark of “C002”.
2. List the maximum, minimum mark for “C021”
3. List the maximum, minimum, average mark for each subject in 5th semester
4. List the name of the courses and average mark of each courses.
5. Calculate the sum of all the scores.
6. How many students are registered for each course? Display the course description and the number of students registered in each course.
7. How many courses did each student register for? Use Assessment table.

OUTPUTS:

1)

```
mysql> select course from course;
ERROR 1054 (42S22): Unknown column 'course' in 'field list'
mysql> select * from course;
+----------+-----------+-------------+--------+
| courseno | regno     | coursedesc  | scores |
+----------+-----------+-------------+--------+
| C001     | 191811306 | mathematics |     93 |
| C001     | 191811307 | mathematics |     80 |
| C002     | 191811306 | OOAD        |     75 |
| C002     | 191811307 | OOAD        |     85 |
| C003     | 191811306 | DBMS        |     91 |
| C003     | 191811307 | DBMS        |     93 |
+----------+-----------+-------------+--------+
6 rows in set (0.00 sec)
```

2)

```
mysql> select courseno from course where scores between 50 and 80;
+----------+
| courseno |
+----------+
| C001     |
| C002     |
+----------+
2 rows in set (0.00 sec)
```

1)

```
mysql> select avg(scores) from course;
+-------------+
| avg(scores) |
+-------------+
|     86.1667 |
+-------------+
1 row in set (0.00 sec)
```

2)

```
mysql> select max(scores),min(scores) from course;
+-------------+-------------+
| max(scores) | min(scores) |
+-------------+-------------+
|          93 |          75 |
+-------------+-------------+
1 row in set (0.00 sec)
```

3)

```
mysql> select max(scores),min(scores),avg(scores) from course;
+-------------+-------------+-------------+
| max(scores) | min(scores) | avg(scores) |
+-------------+-------------+-------------+
|          93 |          75 |     86.1667 |
+-------------+-------------+-------------+
1 row in set (0.00 sec)
```

4)

```
mysql> select avg(scores) from course where courseno like'C00%';
+-------------+
| avg(scores) |
+-------------+
|     68.2000 |
+-------------+
```

5)

```
mysql> select sum(scores) from course;
+-------------+
| sum(scores) |
+-------------+
|         517 |
+-------------+
1 row in set (0.00 sec)
```

6)

```
mysql> select coursedesc,count(coursetype) from course;
+-------------+-------------------+
| coursedesc  | count(coursetype) |
+-------------+-------------------+
| mathematics |                 5 |
+-------------+-------------------+
1 row in set (0.00 sec)
```

7)

```
mysql> select courseno,coursedesc,count(regno) from course where regno like '191811%';
+----------+------------+--------------+
| courseno | coursedesc  | count(regno) |
+----------+------------+--------------+
| C001     | mathematics |            6 |
+----------+------------+--------------+
1 row in set (0.00 sec)
```

RESULT:

The records from the tables are displayed using SELECT commands with WHERE Clause and Pattern matching.

Ex. No.: 6
Date:

SELECT with various clause – GROUP BY, HAVING, ORDER BY

AIM:

To view the records from the tables using SELECT commands with Group By, Having, Order By

DESCRIPTION:

GROUP BY – HAVING:

The GROUP BY clause groups a set of rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row for each group. In other words, it reduces the number of rows in the result set.

The GROUP BY clause is used with aggregate functions such as SUM, AVG, MAX, MIN, and COUNT. The aggregate function that appears in the SELECT clause provides the information about each group.

The GROUP BY clause is an optional clause of the SELECT statement.
To filter the groups returned by GROUP BY clause, you use a HAVING clause.

ORDER BY:

When you use the SELECT statement to query data from a table, the result set is not sorted in any orders. To sort the result set, you use the ORDER BY clause. The ORDER BY clause allows you to:

- Sort a result set by a single column or multiple columns.
- Sort a result set by different columns in ascending or descending order.

SYNTAX:

GROUP BY – HAVING:

```
SELECT
c1, c2,...,cn, aggregate_function(ci)
                                    FROM
                                      table
WHERE
where_conditions
GROUP BY c1 , c2,...,cn
HAVING conditionS
```

Questions:

GROUP BY - HAVING

1. How many students are registered for each course? Display the course description and the number of students registered in each course.
2. How many courses did each student register for? Use Assessment table.

ORDER BY

1. Retrieve Name, Gender, MobileNo of all the students in ascending order of RegNo.
2. List the faculty members in the order of older faculty first.

OUTPUTS:

1)

```
mysql> select courseno,count(courseno) from course group by courseno;
+----------+-----------------+
| courseno | count(courseno) |
+----------+-----------------+
|        1 |               1 |
+----------+-----------------+
1 row in set (0.06 sec)
```

2)

```
mysql> select courseno,count(score) from course group by courseno;
+----------+--------------+
| courseno | count(score) |
+----------+--------------+
|        1 |            1 |
+----------+--------------+
1 row in set (0.00 sec)
```

3)

```
mysql> select name,DOB from student order by regno;
+------+--------+-----------+------+
| sno  | name   | regno     | DOB  |
+------+--------+-----------+------+
|    1 | mukesh | 191811309 | 2001 |
|    2 | nani   | 191811319 | 2001 |
|    3 | ram    | 191811320 | 2000 |
|    4 | ntr    | 191811321 | 2000 |
+------+--------+-----------+------+
```

4)

```
mysql> select * from faculty order by Dob;
+-------+---------+--------+------+-----------+------+
| Facno | FacName | gender | Dob  | mobileno  | DOJ  |
+-------+---------+--------+------+-----------+------+
|     2 | somu    | m      | 1999 | 465415456 | 1987 |
|     1 | ramu    | M      | 2000 |  90515252 | 2001 |
+-------+---------+--------+------+-----------+------+
2 rows in set (0.06 sec)
```

ORDER BY:

SELECT column1, column2,...
FROM tbl
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC],...

ASC stands for ascending and the DESC stands for descending. By default, the ORDER BY clause sorts the result set in ascending order if you don’t specify ASC or DESC explicitly.

RESULT:

The records from the tables are displayed using SELECT commands with GROUP BY, HAVING and ORDER BY.

Ex. No.: 7
Date:

SubQuery& Correlated Query

AIM:

To perform subquery and correlated query on the given relation.

DESCRIPTION:

SUBQUERY

A MySQL subquery is a query nested within another query such as SELECT, INSERT, UPDATE or DELETE. In addition, a MySQL subquery can be nested inside another subquery.

A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.

CORRELATED QUERY:

A correlated subquery is a subquery that uses the data from the outer query. In other words, a correlated subquery depends on the outer query. A correlated subquery is evaluated once for each row in the outer query.

SYNTAX:

SUBQUERY:

```
SELECT
c1, c2,...,cn
                    FROM
                      table
WHERE
c1 IN (SELECT
c1, c2,...,cn
FROM
table WHERE
where_conditions);
```

CORRELATED QUERY:

```
SELECT
*
FROM
table_name
WHERE
EXISTS(subquery );
```

Questions:

Sub-Query and Correlated Sub-Query:

1. Which of the student's score is greater than the highest score?
2. Which of the students' have written more than one assessment test?
3. Which faculty has joined recently and when?
4. List the course and score of assessments that have the value more than the average score each Course

OUTPUTS:

1)

```
mysql> select * from stud2 where marks>(select avg(marks) from stud2);
+-------+------+----------+-------+------------+-----------+
| regno | name | courseno | marks | facultydoj | assements |
+-------+------+----------+-------+------------+-----------+
|  1234 | nani | coo1     |    50 |       2001 |         5 |
|  1254 | ramu | coo1     |    50 |       2002 |         4 |
+-------+------+----------+-------+------------+-----------+
2 rows in set (0.00 sec)
```

2)

```
mysql> select name from stud2 where assements>1;
+------+
| name |
+------+
| nani |
| ramu |
| ravi |
+------+
3 rows in set (0.00 sec)
```

3)

```
mysql> select * from faculty order by doj limit 1;
+-------+-------------+--------+------------+------------+------------+--------+
| FacNo | FacultyName | gender | Dob        | Doj        | Mobileno   | DeptNo |
+-------+-------------+--------+------------+------------+------------+--------+
| fo1   | chaithu     | m      | 0000-00-00 | 0000-00-00 | 2147483647 | cse    |
+-------+-------------+--------+------------+------------+------------+--------+
1 row in set (0.00 sec)
```

4)

```
mysql> select courseno,marks from stud2 where marks>(select avg(marks) from stud2) order by courseno;
+----------+-------+
| courseno | marks |
+----------+-------+
| coo1     |    50 |
| coo1     |    50 |
+----------+-------+
2 rows in set (0.00 sec)
```

RESULT:

The records from the tables are displayed using Sub-Query and Correlated Sub-Query.

Ex. No.: 8
Date:

Joins – EquiJoin, InnerJoin, OuterJoin

AIM:

To perform JOIN using EquiJoin, InnerJoin, OuterJoin on the given relation.

DESCRIPTION:

JOIN

A MySQL join is a method of linking data from one or more table based on values of the common column between tables.

MySQL supports the following types of joins:

1. Cross join
2. Inner join
3. Left join
4. Right join

CROSS JOIN

The CROSS JOIN makes a Cartesian product of rows from multiple tables. Suppose, you join t1 and t2 tables using the CROSS JOIN, the result set will include the combinations of rows from the t1 table with the rows in the t2 table.

INNER JOIN

To join two tables, the INNER JOIN compares each row in the first table with each row in the second table to find pairs of rows that satisfy the join-predicate. Whenever the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of the two tables are included in the result set.

LEFT JOIN

Unlike an INNER JOIN, a LEFT JOIN returns all rows in the left table including rows that satisfy join-predicate and rows do not. For the rows that do not match the join-predicate, NULLs appear in the columns of the right table in the result set.

RIGHT JOIN

A RIGHT JOIN is similar to the LEFT JOIN except that the treatment of tables is reversed. With a RIGHT JOIN, every row from the right table (t2) will appear in the result set. For the rows in the right table that do not have the matching rows in the left table (t1), NULLs appear for columns in the left table (t1).

SYNTAX:

CROSS JOIN:

```
SELECT
t1.id, t2.id
FROM
```

```
t1
CROSS JOIN t2;
```

INNER JOIN:

```
SELECT
t1.id, t2.id
                FROM
                  t1
INNER JOIN
t2 ON t1.pattern = t2.pattern;
```

LEFT JOIN:

```
SELECT
t1.id, t2.id
                FROM
                  t1
LEFT JOIN
t2 ON t1.pattern = t2.pattern
ORDER BY t1.id;
```

RIGHT JOIN:

```
SELECT
t1.id, t2.id
                FROM
                  t1
RIGHT JOIN
t2 on t1.pattern = t2.pattern
ORDER BY t2.id;
```

Questions:

1. List the departments where the faculty members are working.
2. Find the student who has no score in any of the courses. List student name and course number.
3. The office clerk needs the names of the courses taken by the faculty belonging to ‘ECE department’ whose name is ‘Kamal’

OUTPUTS:

1)

```
mysql> select faculty.facno,faculty.facname,department.deptno,department.deptname from faculty cross join department;
+-------+---------+--------+----------+
| facno | facname | deptno | deptname |
+-------+---------+--------+----------+
| 802   | Ratnam  | 11     | Sales    |
+-------+---------+--------+----------+
1 row in set (0.00 sec)
```

2)

```
mysql> select student.name,student.marks,course.courseno from student inner join course on student.course=course.courseno;
+--------+-------+----------+
| name   | marks | courseno |
+--------+-------+----------+
| Ramu   |     0 | C00      |
| Geetha |     0 | C00      |
| Pooja  |     0 | C00      |
+--------+-------+----------+
3 rows in set (0.00 sec)
```

3)

```
mysql> select faculty.facno,faculty.facname,department.deptno,department.deptname from faculty cross join department;
+-------+---------+--------+----------+
| facno | facname | deptno | deptname |
+-------+---------+--------+----------+
| 802   | Ratnam  | 11     | Sales    |
+-------+---------+--------+----------+
1 row in set (0.00 sec)
```

RESULT: The records from the tables are displayed using JOIN using EquiJoinInnerJoin, OuterJoin.

Ex. No.: 9
Date:

VIEW, INDEX, SEQUENCE

AIM:

To create view, index and sequence on the given relation.

DESCRIPTION:

VIEW

MySQL has supported database views since version 5+. In MySQL, almost features of views conform to the SQL: 2003 standard. MySQL processes query against the views in two ways:

1.In a first way, MySQL creates a temporary table based on the view definition
2.statement and executes the incoming query on this temporary table.
3.In a second way, MySQL combines the incoming query with the query defined the view into one query and executes the combined query.

INDEX:

A database index, or just index, helps speed up the retrieval of data from tables. When you query data from a table, first MySQL checks if the indexes exist, then MySQL uses the indexes to select exact physical corresponding rows of the table instead of scanning the whole table..

SEQUENCE:

In MySQL, a sequence is a list of integers generated in the ascending order i.e., 1,2,3… Many applications need sequences to generate unique numbers mainly for identification e.g., customer ID in CRM, employee numbers in HR, equipment numbers in services management system, etc.

To create a sequence in MySQL automatically, you set the AUTO_INCREMENT attribute to a column, which typically is a primary key column.

SYNTAX:

VIEW:

```
CREATE
[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}] VIEW
[database_name].[view_name] AS

[SELECT statement]
```

INDEX:

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
USING [BTREE | HASH | RTREE]
ON table_name (column_name [(length)] [ASC | DESC],...)
```

SEQUENCE:

```
CREATE TABLE table_name(
col_name1 AUTO_INCREMENT PRIMARY
KEY, col_name2,
col_name3, ….);
```

Questions:

Create a view with name ‘std_view’ using STUDENT table which holds the value of register number, name and DOB of student.

Query:-

```
mysql> create view stu_view as select regno,name and dob from stu;
Query OK, 0 rows affected (0.05 sec)

mysql> select * from stu_view;
+-----------+--------------+
| regno     | name and dob |
+-----------+--------------+
| 191811164 |            0 |
| 191811144 |            0 |
|  19181134 |            0 |
|  19181124 |            0 |
|  19181114 |            0 |
+-----------+--------------+
5 rows in set, 5 warnings (0.00 sec)
mysql> create index regno on stu(regno);
Query OK, 0 rows affected (0.36 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show index from stu;
+-------+------------+----------+--------------+-------------+-----------+-----------
-----------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinalit
ex_comment |
+-------+------------+----------+--------------+-------------+-----------+-----------
-----------+
| stu   |          1 | regno    |            1 | regno       | A         |           
          |
+-------+------------+----------+--------------+-------------+-----------+-----------
-----------+
1 row in set (0.00 sec)
```

RESULT:

The records from the tables are displayed using JOIN using EquiJoin,InnerJoin, OuterJoin.

Ex:No: 10
Date:

Simple programming exercise using(REPEAT, WHILE)

Aim:

To learn how to use various MySQL loop statements including while, repeat to run a block of code repeatedly based on a condition.

Procedure:

WHILE loop

The syntax of the WHILE statement is as follows:

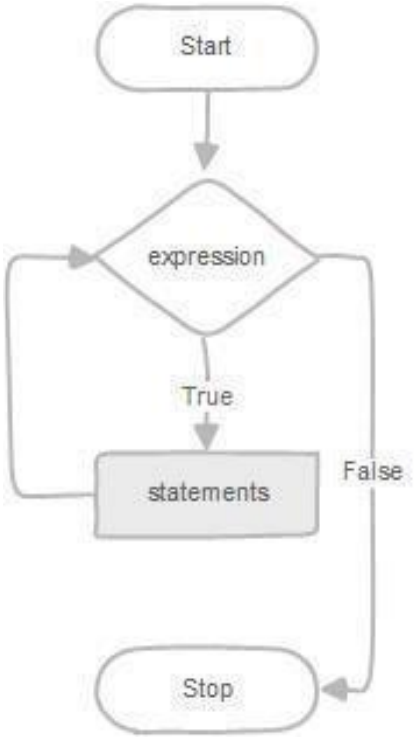
WHILE expression DO

statements

END WHILE

The WHILE loop checks the expression at the beginning of each iteration. If the expression valuates to TRUE, MySQL will execute statements between WHILE and END WHILE until the expression evaluates to FALSE. The WHILE loop is called pretest loop because it checks the expression before the statements execute.

The following flowchart illustrates the WHILE loop statement:



Program1

Write a function to build a string repeatedly until the value of the variable becomes s greater than 5.
Then, we display the final string using a SELECT statement.

Procedure:

REPEAT loop

The syntax of the REPEAT loop statement is as follows:

REPEAT

statements;

UNTIL expression

END REPEAT

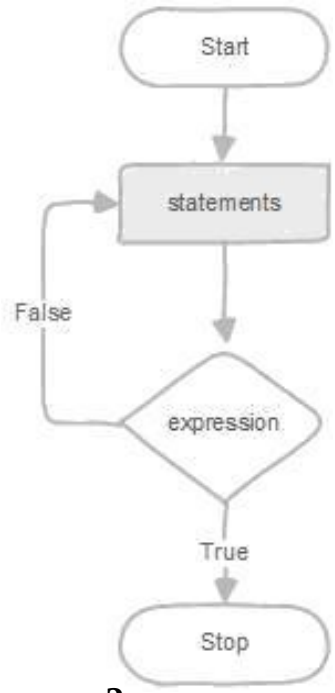
First, MySQL executes the statements, and then it evaluates the expression. If

the expression evaluates to FALSE, MySQL executes the statements repeatedly until

the expression evaluates to TRUE.

Because the REPEAT loop statement checks the expression after the execution of statements, the REPEAT loop statement is also known as the post-test loop.

The following flowchart illustrates the REPEAT loop statement:



Program 2:

Write a function that uses REPEAT statement which would repeat the loopuntil income is greater than or equal to 4000, at which point the REPEAT loop would beterminated

Program :-1

```
mysql> CREATE PROCEDURE test_mysql_while_loop()
    ->  BEGIN
    ->  DECLARE x  INT;
    ->  DECLARE str  VARCHAR(255);
    ->
    ->  SET x = 1;
    ->  SET str =  '';
    ->
    ->  WHILE x  <= 5 DO
    ->  SET  str = CONCAT(str,x,',');
    ->  SET  x = x + 1;
    ->  END WHILE;
    ->
    ->  SELECT str;
    ->  END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL test_mysql_while_loop() //
+------------+
| str        |
+------------+
| 1,2,3,4,5, |
+------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Program :-2

```
mysql> CREATE PROCEDURE dorepeat(p1 INT) BEGIN SET @x=0; REPEAT SET @x=@x+1; UNT
IL @x>p1 END REPEAT; END//
Query OK, 0 rows affected (0.01 sec)

mysql> CALL dorepeat(4001) //
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT @x;
    -> //
+------+
| @x   |
+------+
| 4002 |
+------+
1 row in set (0.00 sec)
```

RESULT: Thus the Simple programming exercise using(REPEAT, WHILE) exectuted successfully.

Ex:No: 11
Date:

Simple programming exercise using(CASE and LOOP)

Aim:

To learn how to use various MySQL loop statements including case and loop to run a block of code repeatedly based on a condition.

Procedure:

In MySQL, the CASE statement has the functionality of an IF-THEN-ELSE statement and has 2 syntaxes that we will explore.

CASE Syntax
CASE case_value
WHEN when_value THEN statement_list
[WHEN when_value THEN statement_list] ...
[ELSE statement_list]
END CASE

Program 1:

Write a function that uses CASE statement where if monthly_value is equal to or less than 4000, then income_level will be set to 'Low Income'. If monthly_value is equal to or less than 5000, then income_level will be set to 'Avg Income'. Otherwise, income_levelwill be set to 'High Income'.

Procedure:

LOOP Syntax

[begin_label:] LOOP

statement_list

END LOOP [end_label]

LOOP implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon

(;) statement delimiter. The statements within the loop are repeated until the loop is terminated. Usually, this is accomplished with a LEAVE statement. Within a stored function, RETURN can also be used, which exits the function entirely.

Program 2:

Write a function that will use ITERATE statement which would cause the loop to repeat while income is less than 4000. Once income is greater than or equal to 4000, would terminate the LOOP.

1)

```
mysql> CREATE FUNCTION INCOMELEVEL(MONTHLY_VALUE INT) RETURNS VARCHAR(20) BEGIN
DECLARE INCOME_LEVEL VARCHAR(20); CASE MONTHLY_VALUE WHEN 4000 THEN SET INCOME_L
EVEL='LOW INCOME'; WHEN 5000 THEN SET INCOME_LEVEL='AVG INCOME'; ELSE SET INCOME
_LEVEL='HIGH INCOME'; RETURN INCOME_LEVEL; END; //
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near '' at
line 1
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION IncomeLevel ( monthly_value INT )
    -> RETURNS varchar(20)
    ->
    -> BEGIN
    ->
    ->    DECLARE income_level varchar(20);
    ->
    ->    CASE monthly_value
    ->       WHEN 4000 THEN
    ->         SET income_level = 'Low Income';
    ->
    ->       WHEN 5000 THEN
    ->         SET income_level = 'Avg Income';
    ->
    ->       ELSE
    ->         SET income_level = 'High Income';
    ->    END CASE;
    ->
    ->    RETURN income_level;
    ->
    -> END; //
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT INCOMELEVEL(5300); //
+------------------+
| INCOMELEVEL(5300) |
+------------------+
| High Income       |
+------------------+
1 row in set (0.00 sec)

mysql>
```

2)

```
mysql> SELECT INCOMELEVEL2(2100);
    -> //
ERROR 1054 (42S22): Unknown column 'starting_value' in 'field list'
mysql> SELECT IncomeLevel2(2100);
    -> //
ERROR 1054 (42S22): Unknown column 'starting_value' in 'field list'
mysql> DROP FUNCTION INCOMELEVEL2 //
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE FUNCTION CALCINCOME2 ( starting_value INT )
    -> RETURNS INT
    ->
    -> BEGIN
    ->
    ->    DECLARE income INT;
    ->
    ->    SET income = 0;
    ->
    ->    label1: LOOP
    ->      SET income = income + starting_value;
    ->      IF income < 4000 THEN
    ->        ITERATE label1;
    ->      END IF;
    ->      LEAVE label1;
    ->    END LOOP label1;
    ->
    ->    RETURN income;
    ->
    -> END; //
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CALCINCOME2(2100);
    -> //
+------------------+
| CALCINCOME2(2100) |
+------------------+
|              4200 |
+------------------+
1 row in set (0.00 sec)

mysql>
```

RESULT: Thus the Simple programming exercise using(CASE and LOOP) excuted successfully.

Ex:No: 12
Date:

TCL COMMANADS

Aim:

To learn how to use various TCL commands Commit, Rollback and Savepoint
SQL commands

Procedure and Syntax:

Transaction Control Language(TCL) commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

COMMIT command

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent

SYNTAX;

COMMIT;

ROLLBACK command

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Syntax:

```
ROLLBACK TO savepoint_name;
```

SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Syntax:

```
SAVEPOINT savepoint_name;
```

Problem 1:

Rollback to that state using the ROLLBACK command whenever required.
Create a following table Class and insert values into it in the order and create savepoints in between them. Try to rollback t the save point and check your output by giving select commands.

Let us use some SQL queries on the above table and see the results.

```
INSERTINTO class VALUES(5,'Rahul');

COMMIT;

UPDATE class SET name ='Abhijit' WHERE id ='5';

SAVEPOINT A;

INSERTINTO class VALUES(6,'Chris');

SAVEPOINT B;

INSERTINTO class VALUES(7,'Bravo');

SAVEPOINT C;
```

The resultant table will look like,

Now let's use the ROLLBACK command to roll back the state of data to the savepoint B.

```
ROLLBACKTO B;

SELECT*FROM class;
```

Now our class table will look like,

Now let's again use the ROLLBACK command to roll back the state of data to the savepoint A

```
ROLLBACKTO A;

SELECT*FROM class;
```

Now the table will look like,

Questions:-

```
mysql> create table clss(name varchar(10),id int(5));
Query OK, 0 rows affected (0.19 sec)

mysql> insert into clss values("dj",5);
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.04 sec)

mysql> update class set name="bravo" where id="5";
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> savepoint A;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into clss values("uppal",6);
Query OK, 1 row affected (0.00 sec)

mysql> savepoint B;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into clss values("balu",7);
Query OK, 1 row affected (0.00 sec)

mysql> savepoint C;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from clss;
+-------+------+
| name  | id   |
+-------+------+
| dj    |    5 |
| uppal |    6 |
| balu  |    7 |
+-------+------+
3 rows in set (0.00 sec)

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

mysql> select * from clss;
+-------+------+
| name  | id   |
+-------+------+
| dj    |    5 |
| uppal |    6 |
+-------+------+
2 rows in set (0.00 sec)

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

mysql> select * from clss;
+------+------+
| name | id   |
+------+------+
| dj   |    5 |
+------+------+
1 row in set (0.00 sec)
```

Result:

So now we know how the commands COMMIT, ROLLBACK and SAVEPOINT works.

Ex:No: 13
Date:

DCL COMMANADS

Aim:

To learn how to use various DCL commands GRANT and REVOKE SQL commands

Procedure and Syntax:

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.

Allow a User to create session

When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/privileges are granted to the user.

Following command can be used to grant the session creating privileges.

```
GRANT CREATE SESSION TO username;
```

Allow a User to create table

To allow a user to create tables in the database, we can use the below command,

```
GRANT CREATE TABLE TO username;
```

Provide user with space on tablespace to store table

Allowing a user to create table is not enough to start storing data in that table. We also must provide the user with privileges to use the available tablespace for their table and data.

```
ALTER USER username QUOTA UNLIMITED ON SYSTEM;
```

The above command will alter the user details and will provide it access to unlimited tablespace on system.

NOTE: Generally unlimited quota is provided to Admin users.

Grant all privilege to a User

sysdbais a set of priviliges which has all the permissions in it. So if we want toprovide all the privileges to any user, we can simply grant them the sysdba permission.

```
GRANT sysdba TO username
```

Grant permission to create any table

Sometimes user is restricted from creating come tables with names which are reserved for system tables. But we can grant privileges to a user to create any table using the below command,

```
GRANT CREATE ANY TABLE TO username
```

Grant permission to drop any table

As the title suggests, if you want to allow user to drop any table from the database, then grant this privilege to the user,

```
GRANT DROP ANY TABLE TO username
```

To take back Permissions

And, if you want to take back the privileges from any user, use the REVOKE command.

```
REVOKE CREATE TABLE FROM username
```

RESULT: Thus the DCL commands GRANT and REVOKE SQL executed successfully.

Ex:No: 14
Date:

HIGH LEVEL PROGRAMMING EXTENSIONS (PROCEDURES)

Aim:

To implement procedures using program in MySQL.

PROCEDURES:

A procedure is a subprogram that performs a specific action.

Creating a procedure

We use the CREATE PROCEDURE statement to create a new stored procedure. We specify the name of stored procedure after the CREATE PROCEDURE statement. The DELIMITER command is used to change the

standard delimiter of MySQLcommands (i.e. ;). As the statements within the routines (functions, stored procedures or triggers) end with a semi-colon (;), to treat them as a compound statement we use DELIMITER.

Calling stored procedures(Executing a procedure)

In order to call a stored procedure, you use the following SQL command:

CALL stored_procedure_name();

Program 1:

Create a simple procedure to get all the records from the table ‘student_info’ which have the following data:

```
mysql> select * from student_info;
+----- +---------+------------ +------------  +
| id     | Name| Address   | Subject      |
+----- +---------+------------ +------------  +
| 100 | Aarav  | Delhi       | Computers |
| 101 | YashPal | Amritsar | History     |
| 105 | Gaurav | Jaipur      | Literature |
| 110 | Rahul  | Chandigarh | History   |
+------  +--------+------------ +------------  +
```

Program 2:

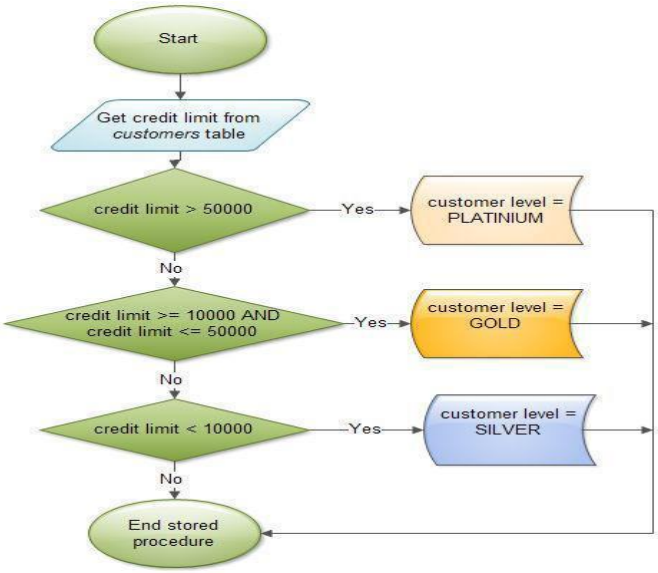
Create a stored procedure GetCustomerLevel() that accepts two parameters customer number and customer level.

- First, it gets the credit limit from the customers table.

Then, based on the credit limit, it determines the customer level: PLATINUM , GOLD , an SILVER.

The parameter p_customerlevel stores the level of the customer and is used by the calling program.

The following flowchart demonstrates the logic of determining customer level.



The table ‘customers’ should have the following attributes:

customers(cno , cname, creditlimit)

Program :-1

```
mysql> CREATE PROCEDURE student_info()
    -> select * from student_info;
Query OK, 0 rows affected (0.00 sec)

mysql> call student_info();
+-------+-------+-----------+---------+
| stuid | name  | area      | subject |
+-------+-------+-----------+---------+
|   201 | raj   | chennai   | dbms    |
|   202 | rahul | hyderabad | ooad    |
|   203 | rahim | munbai    | java    |
|   204 | vikas | kochi     | python  |
+-------+-------+-----------+---------+
4 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Program :-2

```
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE GetCustomerLevel(
    ->     in  p_customerNumber int(11),
    ->     out p_customerLevel  varchar(10))
    -> BEGIN
    ->     DECLARE creditlim double;
    ->
    ->     SELECT creditlimit INTO creditlim
    ->     FROM customers
    ->     WHERE customerNumber = p_customerNumber;
    ->
    ->     IF creditlim > 50000 THEN
    ->  SET p_customerLevel = 'PLATINUM';
    ->     ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
    ->         SET p_customerLevel = 'GOLD';
    ->     ELSEIF creditlim < 10000 THEN
    ->         SET p_customerLevel = 'SILVER';
    ->     END IF;
    ->
    -> END$$
Query OK, 0 rows affected (0.00 sec)
```

RESULT: Thus the program in MySQL executed successfully.

Ex:No: 15
Date:

HIGH LEVEL PROGRAMMING EXTENSIONS (FUNCTIONS)

Aim:

To implement Functions using program in MySQL.

FUNCTIONS:

A function is a subprogram that computes a value.

Creating a function

The CREATE FUNCTION statement is also used in MySQL to support UDFs (user-defined functions). A UDF can be regarded as an external stored function.

MySQL stored function syntax

CREATE FUNCTION function_name(param1,param2,…)
RETURNS datatype
[NOT] DETERMINISTIC
statements

Program 1:

Create a function that returns the level of a customer based on credit limit.(Use the IF statement to determine the credit limit).

The table 'customers' should have the following attributes:

customers(cno , cname, creditlimit)

If credit limit > 50000 then customer_level = PLATINUM

If credit limit >= 10000 AND credit limit <= 50000 then customer_level = GOLD

If credit limit credit limit < 10000 then customer_level = SILVER

RECURSION in Mysql Procedures

Mysql version should be >= 5.

Have to set system parameters. This means putting the recursion count limit.
SET @@GLOBAL.max_sp_recursion_depth = 255;
SET @@session.max_sp_recursion_depth = 255;

Program 2

Write a recursive MySQL procedure compute the factorial of a number .

OUTPUT;

1

```
mysql> DELIMITER //
mysql> CREATE FUNCTION CustomerLevel(p_CREDITLIMIT INT) RETURNS VARCHAR(10)
    -> DETERMINISTIC
    -> BEGIN
    -> DECLARE lvl VARCHAR(10);
    -> IF p_CREDITLIMIT > 50000 THEN
    -> SET lvl = 'PLATINUM';
    -> ELSEIF (p_CREDITLIMIT <= 50000 AND p_CREDITLIMIT >= 10000) THEN
    -> SET lvl = 'GOLD';
    -> ELSEIF p_CREDITLIMIT < 10000 THEN
    -> SET lvl = 'SILVER';
    -> END IF;
    -> RETURN (lvl);
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT NAME,CustomerLevel(CREDITLIMIT)
    -> FROM CUSTOMER
    -> ORDER BY NAME
    -> //
ERROR 1054 (42S22): Unknown column 'NAME' in 'field list'
mysql> SELECT CNAME,CustomerLevel(CREDITLIMIT)
    -> FROM CUSTOMER
    -> ORDER BY NAME
    -> //
ERROR 1054 (42S22): Unknown column 'NAME' in 'order clause'
mysql> SELECT CNAME,CustomerLevel(CREDITLIMIT) FROM CUSTOMER ORDER BY CNAME//
+----------+---------------------------+
| CNAME    | CustomerLevel(CREDITLIMIT) |
+----------+---------------------------+
| DINESH   | GOLD                      |
| NAGENDRA | PLATINUM                  |
| RAJA     | GOLD                      |
| RAMU     | SILVER                    |
+----------+---------------------------+
4 rows in set (0.00 sec)
```

2)

```
mysql> DELIMITER //
mysql> CREATE FUNCTION CustomerLevel(p_CREDITLIMIT INT) RETURNS VARCHAR(10)
    -> DETERMINISTIC
    -> BEGIN
    -> DECLARE lvl VARCHAR(10);
    -> IF p_CREDITLIMIT > 50000 THEN
    -> SET lvl = 'PLATINUM';
    -> ELSEIF (p_CREDITLIMIT <= 50000 AND p_CREDITLIMIT >= 10000) THEN
    -> SET lvl = 'GOLD';
    -> ELSEIF p_CREDITLIMIT < 10000 THEN
    -> SET lvl = 'SILVER';
    -> END IF;
    -> RETURN (lvl);
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT NAME,CustomerLevel(CREDITLIMIT)
    -> FROM CUSTOMER
    -> ORDER BY NAME
    -> //
ERROR 1054 (42S22): Unknown column 'NAME' in 'field list'
mysql> SELECT CNAME,CustomerLevel(CREDITLIMIT)
    -> FROM CUSTOMER
    -> ORDER BY NAME
    -> //
ERROR 1054 (42S22): Unknown column 'NAME' in 'order clause'
mysql> SELECT CNAME,CustomerLevel(CREDITLIMIT) FROM CUSTOMER ORDER BY CNAME//
+----------+----------------------------+
| CNAME    | CustomerLevel(CREDITLIMIT) |
+----------+----------------------------+
| DINESH   | GOLD                       |
| NAGENDRA | PLATINUM                   |
| RAJA     | GOLD                       |
| RAMU     | SILVER                     |
+----------+----------------------------+
4 rows in set (0.00 sec)
```

Progarm:-1

```
mysql> DELIMITER //
mysql> CREATE FUNCTION customerlevel(p_CREDITLIMIT INT)RETURNS VARCHAR(10)
    -> DETERMINISTIC
    -> BEGIN
    -> DECLARE lvl varchar(10);
    -> IF p_CREDITLIMIT > 50000 THEN
    -> SET lvl='PLATINUM';
    -> ELSEIF(p_CREDITLIMIT <=50000 AND p_CREDITLIMIT>=10000)THEN
    -> SET lvl ='GOLD';
    -> ELSEIF  p_CREDITLIMIT < 10000 THEN
    -> SET lvl ='SILVER';
    -> END IF;
    -> RETURN (lvl);
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT NAME,customerLevel(CREDITLIMIT)
    -> FROM CUSTOMER
    -> ORDER BY NAME //
```

Progarm:-2

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE find_fact(IN n INT)
    -> BEGIN
    ->  SET @@GLOBAL.max_sp_recursion_depth=255;
    ->  SET @@session.max_sp_recursion_depth=255;
    ->  CALL factorial(n,@fact);
    ->  SELECT @fact;
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER $$
mysql> CREATE PROCEDURE factorial(IN n INT,OUT fact INT)
    -> BEGIN
    ->  IF n=1 THEN
    ->     SET fact:=1;
    ->  ELSE
    ->    CALL factorial(n-1,fact);
    ->    SET fact:=n*fact;
    ->   END IF;
    -> END
    -> $$
Query OK, 0 rows affected (0.01 sec)

mysql> CALL find_fact(5);
    -> $$
+--------+
| @fact  |
+--------+
|   120  |
+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
```

RESULT: Thus the Functions using program in MySQL executed successfully .

Ex.No: 16
Date:

HIGH LEVEL LANGUAGE EXTENSION WITH CURSORS

Program 1

Write a Cursor program using MySQL to retrieve the email-ids(build an email list) of employees from employees table.

SOLUTION :

```
create table employees(id integer, Name varchar(100), email varchar(100)); insert into employees(id,
Name, email) values(1, "Harry Potter", "pharry@warnerbros.com");
insert into employees(id, Name, email) values(2, "Clark Kent",
"kclark@dccomics.com");
insert into employees(id, Name, email) values(3, "Tony Stark", "stony@marvel.com");


DELIMITER $$
CREATE PROCEDURE build_email_list (INOUT email_listvarchar(4000))
BEGIN
DECLARE v_finished INTEGER DEFAULT 0;
DECLARE v_emailvarchar(100) DEFAULT "";

-- declare cursor for employee email

DEClAREemail_cursor CURSOR FOR
SELECT email FROM employees;

-- declare NOT FOUND handler
DECLARE CONTINUE HANDLER FOR
NOT FOUND SET v_finished = 1;
OPEN email_cursor;
get_email: LOOP
FETCH email_cursor INTO v_email;
IF v_finished = 1 THEN
LEAVE get_email;
END IF;

-- build email list

SET email_list = CONCAT(v_email,";",email_list);
END LOOP get_email;
CLOSE email_cursor;
END$$
DELIMITER ;
```

-- Calling the procedure and getting the email list

```
SET @email_list = "";
CALL build_email_list(@email_list);
SELECT @email_list;
```

Program :-1

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE build_email_list (INOUT email_list varchar(4000))
    -> BEGIN
    -> DECLARE v_finished INTEGER DEFAULT 0;
    ->     DECLARE v_email varchar(100) DEFAULT "";
    -> DECLARE email_cursor CURSOR FOR
    -> SELECT email FROM employees;
    -> DECLARE CONTINUE HANDLER FOR
    ->     NOT FOUND SET v_finished = 1;
    -> OPEN email_cursor;
    -> get_email:LOOP
    -> FETCH email_cursor INTO v_email;
    -> IF v_finished = 1 THEN
    -> LEAVE get_email;
    -> END IF;
    -> SET email_list = CONCAT(v_email,";",email_list);
    -> END LOOP get_email;
    -> CLOSE email_cursor;
    -> END $$
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> SET @email_list = "";
Query OK, 0 rows affected (0.00 sec)

mysql> CALL build_email_list(@email_list);
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> select @email_list;
+-------------------------------------------------------------+
| @email_list                                                 |
+-------------------------------------------------------------+
| stony@marvel.com;kclark@deconics.com;pharry@warnerbros.com; |
+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

RESULT:

stony@marvel.com;kclark@dccomics.com;pharry@warnerbros.com;

Ex:No:17
Date:

TRIGGER

Aim:

To implement trigger in MySQL.

A trigger or database trigger is a stored program executed automatically to respond to a specific event e.g., insert, update or delete occurred in a table.

Create trigger syntax

CREATE TRIGGER trigger_nametrigger_timetrigger_event
ON table_name
FOR EACH ROW
BEGIN
...
END;

Program 1 :

Create a trigger in MySQL to log the changes of the EMPLOYEES table with fields ID, Name and Email. Also create a new table named EMPLOYEES_AUDIT to keep the changes of the employee table. Create a BEFORE UPDATE trigger that is invoked before a change is made to the employees table.

Program:-

```
mysql> DELIMITER //
mysql> CREATE TRIGGER before_student_update
    -> BEFORE UPDATE ON student
    -> FOR EACH ROW
    -> BEGIN
    -> INSERT INTO student_audit
    -> SET action ='update',
    -> student_id =OLD.id,
    -> lastname=OLD.Name,
    -> changedat = NOW();
    -> END //
Query OK, 0 rows affected (0.06 sec)

mysql> DELIMITER;
mysql> update student set name ='tony stark_c'Where id=3;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> select * from student_audit;
+----+------------+------------+---------------------+--------+
| id | student_id | lastname   | changedat           | action |
+----+------------+------------+---------------------+--------+
|  1 |          3 | tony stark | 2019-08-12 13:07:44 | update |
+----+------------+------------+---------------------+--------+
1 row in set (0.00 sec)
```

RESULT: Thus the trigger is executed successfully.

Exp No:18

Date:

DATABASECONNECTIVITYUSINGPHP AND MYSQL

AIM:

ToconnecttheMysqlusingPHPandMYSQLandtoexecutetheCREATE,INSERT,SELECTcommandin MySQL

PROGRAM:

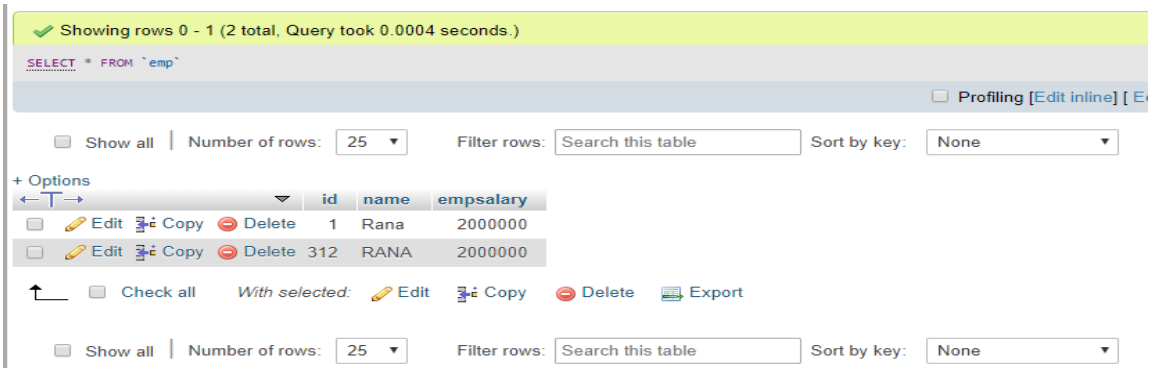
```
<?php

$host=

$password="";

$conn=mysqli_connect($host,$user,$password);

if(!$conn)

{

Die('couldnot connect:',mysql_connect_error());

}

echo"connectsuccessfully('br/>')";

$sql='Create database mydb';

$sql="create table emp(id int,namevarchar(10) NOT NULL,empsalary INT NOT NULL,primary key(id))";

$sql="insert into emp(id,name,empsalary) values(312,RANA,200000)";

$sql="delete from emp where id=1";

$sql="updateemp set empsalary=9000000 where id=312";

if(mysqli_query($conn,$sql))

{echo "operations failed failed",mysqli_error($conn);
```

```
}

mysqli_close($conn);

?>
```

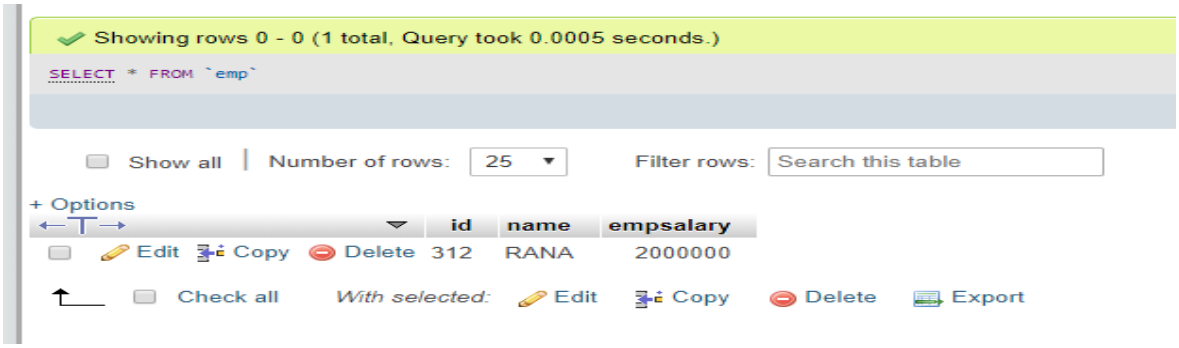
OUTPUT:

After creation of database and emp table

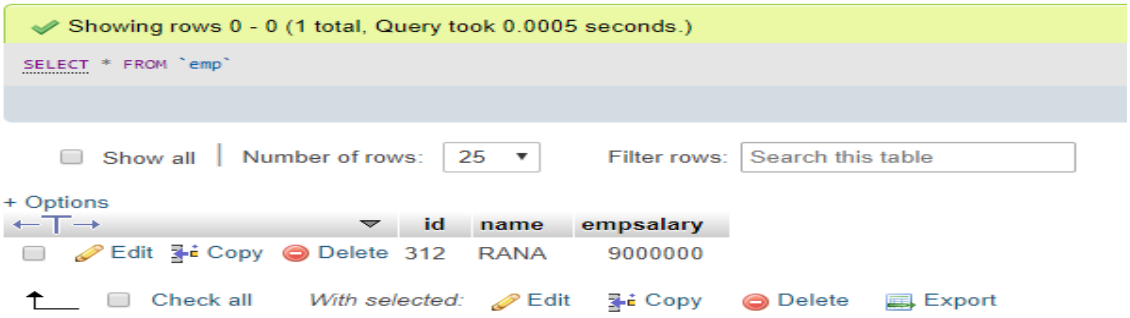
After insertion:



After deletion:



After updation:



RESULT: ThustheMysqlconnectedusingPHP and MYSQLandexecutedtheCREATE,INSERT,SELECTcommandinMySQL.