Experiment No: 1

Aim: - **Introduction to MySQL, Database Creation, Table Creation.**

- **Create a new database (Select an example of your own wish)**
- **View all databases • Learn about data types of attributes and create a new table in the database with different data types**
- **Use of alter command to add and drop attributes**
- **Modify attribute name**
- **Use of desc command to display information about a table**
- **Rename a table • Create more tables for your database**

Theory: -

What is SQL?

The structured query language is used to communicate with several relational database such as MySQL, Oracle and Cassandra.

SQL Datatypes:

CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum string length in characters - can be from 0 to 6553
INT(size)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)

Data Definition Language(DDL): It consists of a set of commands for defining relation schema, deleting relations, and modifying relation schemas.

Syntax
CREATE DATABASE database_name;
Example
CREATE DATABASE 2 _DB;

```
SHOW DATABASES;
USE DATABASE 2 _DB
```

The table can be created inside the database as follows -
CREATE TABLE table name (
Col1_name datatype,
col2 _name datatype,
……
coln_name datatype
);

Alteration

There are SQL command for alteration of table. That means we can add new column or delete some column from the table using these alteration commands

Syntax for Adding columns

ALTER TABLE table_name

ADD column_name datatype;

SQL QUERY: -

```
mysql> create database pratical;
Query OK, 1 row affected (0.05 sec)

mysql>
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| 20_db              |
| god                |
| info               |
| information_schema |
| mysql              |
| performance_schema |
| pratical           |
| sys                |
+--------------------+
8 rows in set (0.00 sec)

mysql>
```

```
mysql> create table student(
    -> Name varChar(10),
    -> Roll_no int,
    -> class varChar(3),
    -> primary key(Roll_no)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql>
```

```
mysql> alter table student
    -> add gender varChar(1);
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table student
    -> drop column class;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

```
mysql> alter table student
    -> rename column Roll_no to Reg_id;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

```
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| Name   | varchar(10) | YES  |     | NULL    |       |
| Reg_id | int         | NO   | PRI | NULL    |       |
| gender | varchar(1)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> describe student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| Name   | varchar(10) | YES  |     | NULL    |       |
| Reg_id | int         | NO   | PRI | NULL    |       |
| gender | varchar(1)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Conclusion: -

Successfully created database and table using SQL Query.

Experiment No: 2

Aim: - Data insertion, update/modification/Delete and retrieval through MySQL. Basic SQL structure. Query implementation

- **Insert tuples in the table including null values in the tuple**
- **Update values in the table**
- **Delete tuples in the table**
- **Query to view all tuples of the table**
- **Run basic queries to view particular attributes of a table**
- **Run basic queries to use basic comparison operators**
- **Run basic queries to view find certain tuples of a table**
- **Run queries using order by, limit operators**

Theory: -

Data Manipulation Language(DML): It consists of set of SQL commands for inserting tuples into relational schema, deleting tuples from or modifying tuples in databases.

The UPDATE statement is used to modify the existing records in a table.

Syntax: UPDATE *table_name*
SET *column1* = *value1*, *column2* = *value2*, ...
WHERE *condition*;

The INSERT INTO statement is used to insert new records in a table.

Syntax: INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

The DELETE statement is used to delete existing records in a table.

Syntax: DELETE FROM *table_name* WHERE *condition*;

The WHERE clause is used to filter records.

Syntax: SELECT * FROM Customers
WHERE Country='Mexico';

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

Syntax: SELECT * FROM Customers
ORDER BY CustomerName;

SQL Query: -

```
mysql> use Comp;
Database changed
mysql> create table Employee(
    -> name varchar(50),
    -> occupation varchar(40),
    -> working_date date,
    -> working_hours varchar(10)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql>
```

```
mysql> Insert into emp
    -> values(
    -> "Harsh",
    -> "Data Analyst",
    -> "2024-05-01",
    -> 9
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> select * from emp;
+-------+-------------+-------------+--------------+
| name  | occupation  | working_date | working_hour |
+-------+-------------+-------------+--------------+
| Harsh | Data Analyst | 2024-05-01  |            9 |
+-------+-------------+-------------+--------------+
1 row in set (0.00 sec)
```

```
mysql> CREATE TABLE Orders (
    ->     OrderID int NOT NULL,
    ->     Rol_no int NOT NULL,
    ->     PRIMARY KEY (OrderID),
    ->     FOREIGN KEY (Rol_no) REFERENCES CSE_AIML(Rol_no)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql>
```

```
mysql> update Emp
    -> set name = "Amit",
    -> working_hour = 10
    -> where occupation = "Data Analyst";
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from Emp;
+---------+-------------+--------------+--------------+
| name    | occupation  | working_date | working_hour |
+---------+-------------+--------------+--------------+
| Amit    | Data Analyst | 2024-05-01   |           10 |
| Abhishak | Programmer  | 2020-05-22   |           16 |
+---------+-------------+--------------+--------------+
2 rows in set (0.00 sec)
```

Conclusion: - Successfully implemented Data Insertion, update/modification/delete and retrieval through MySQL

Experiment No: 3

AIM: - Enforcing integrating constraints (domain, key constraints (Primary/Foreign key), Not null, Unique, Default, Check).

- **Create a table with appropriate primary key**
- **Alter a table to add primary key**
- **Drop a primary key**
- **Add a foreign key while create a table**
- **Alter table to add a foreign key**
- **Drop a foreign key**
- **Include constraints like null/not noll, unique, default, check**
- **Drop a constraint**

Theory: -

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

Syntax:
```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

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

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

Syntax:
```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

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

The NOT NULL constraint enforces a column to NOT accept NULL values.

Syntax:
```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

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.

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

Syntax:
```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
```

```
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

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.

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.

SQL Query: -

```
mysql> create table student(
    -> Name varChar(10),
    -> Roll_no int,
    -> class varChar(3),
    -> primary key(Roll_no)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql>
```

```
mysql> alter table CSE_AIML
    -> add primary key(Roll_no);
```

```
mysql> alter table CSE_AIML
    -> drop primary key;
Query OK, 0 rows affected (0.16 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

```
mysql> CREATE TABLE Orders (
    ->     OrderID int NOT NULL,
    ->     Rol_no int NOT NULL,
    ->     PRIMARY KEY (OrderID),
    ->     FOREIGN KEY (Rol_no) REFERENCES CSE_AIML(Rol_no)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql>
```

```
mysql> create table canteen(
    -> Name varChar(20) Not Null,
    -> RegNo int Unique,
    -> desg varChar(10) DEFAULT 'STUDENT'
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql>
```

```
mysql> ALTER TABLE CANTEEN
    -> DROP CONSTRAINT RegNo;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

Conclusion: - Successfully Enforced integrating constraints (domain, key constraints (Primary/Foreign key), Not null, Unique, Default, Check).

Experiment no: 4

Aim: - **Creating and updating view, query implementation using view.**

- **Learn about the objective of using views**
- **Learn to create views**
- **Insert, delete, update data in views**
- **Run queries on views**

Theory: -

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

Syntax: CREATE VIEW *view_name* AS
SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;

A view can be updated with the CREATE OR REPLACE VIEW statement.

A view is deleted with the DROP VIEW statement.

Syntax : DROP VIEW *view_name*;

SQL Query: -

```
mysql> create view trainer as
    -> select name, subject
    -> from student;
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> alter view trainer as
    -> select name, student_id
    -> from student;
Query OK, 0 rows affected (0.02 sec)

mysql> select * from trainer;
+---------+------------+
| name    | student_id |
+---------+------------+
| sujal   |          1 |
| avadesh |          2 |
+---------+------------+
2 rows in set (0.00 sec)
```

Conclusion: - Successfully Created and updated view, query implementation using view.

Experiment No: - 5

Aim: - **Use of aggregate functions(AVG, COUNT, MIN, MAX, SUM).**

- **Learn about aggregate functions**
- **Run queries to find sum, average, count, count-distinct, minimum, maximum**
- **Run queries using aggregate function with null values**

Theory: -

Aggregate Functions:

The COUNT() function returns the number of rows that matches a specified criterion.

Syntax: SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

The AVG() function returns the average value of a numeric column.

Syntax: SELECT AVG(*column_name*)
FROM *table_name*
WHERE *condition*;

The SUM() function returns the total sum of a numeric column.

Syntax: SELECT SUM(*column_name*)
FROM *table_name*
WHERE *condition*;

The MIN() function returns the smallest value of the selected column.

Syntax: SELECT MIN(Price)
FROM Products;

The MAX() function returns the largest value of the selected column.

Syntax: SELECT MAX(Price)
FROM Products;

SQL Query: -

```
mysql> select * from employee;
+---------+-----------+--------------+---------------+-------+
| name    | occupation | working_date | working_hours | sal   |
+---------+-----------+--------------+---------------+-------+
| robin   | scientist  | 2020-10-04   | 12            | 10000 |
| warner  | enginner   | 2020-10-04   | 10            | 23420 |
| Marco   | Doctor     | 2020-10-04   | 14            | 10000 |
| Brayden | teacher    | 2020-10-04   | 12            | 23420 |
| Antonio | business   | 2020-10-04   | 11            | 30000 |
| Peter   | actor      | 2020-10-04   | 13            | 30000 |
+---------+-----------+--------------+---------------+-------+
6 rows in set (0.01 sec)

mysql> select sum(sal)
    -> from employee;
+----------+
| sum(sal) |
+----------+
|   126840 |
+----------+
1 row in set (0.03 sec)

mysql> select avg(sal)
    -> from employee;
+------------+
| avg(sal)   |
+------------+
| 21140.0000 |
+------------+
1 row in set (0.00 sec)
```

```
mysql> select count(sal)
    -> from employee;
+------------+
| count(sal) |
+------------+
|          6 |
+------------+
1 row in set (0.00 sec)

mysql> select COUNT(DISTINCT sal)
    -> from employee;
+----------------------+
| COUNT(DISTINCT sal) |
+----------------------+
|                    3 |
+----------------------+
1 row in set (0.00 sec)

mysql> select max(sal)
    -> from employee;
+----------+
| max(sal) |
+----------+
|    30000 |
+----------+
1 row in set (0.00 sec)

mysql> select min(sal)
    -> from employee;
+----------+
| min(sal) |
+----------+
|    10000 |
+----------+
```

Conclusion: - Successfully implemented the use of aggregate functions(AVG, COUNT, MIN, MAX, SUM).

Experiment 6

Use of Join operator (Natural join, Outer join (left, right and full)

• Run queries to find natural join, join, outer join, right join, left join of two or more tables

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

"Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

"Customers" table:

CustomerID	CustomerName	Country
1	Alfreds	Germany
2	Ana Trujillo	Mexico
3	Antonio Moreno	Mexico

Inner Join

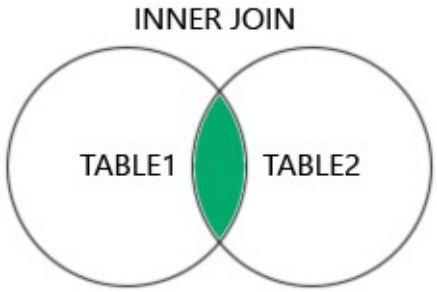
```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Output :

OrderID	CustomerName	OrderDate

Types of Joins in MySQL

- INNER JOIN: Returns records that have matching values in both tables
- LEFT JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT JOIN: Returns all records from the right table, and the matched records from the left table



INNER JOIN Syntax

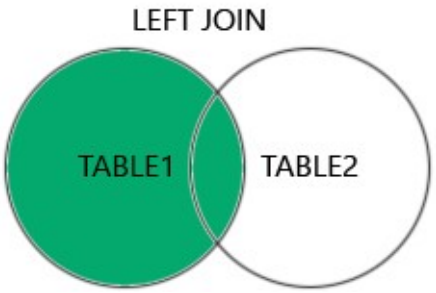
```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Example

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).



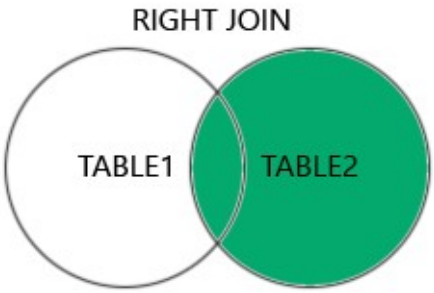
LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).



RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

SQl Query:

```
mysql>  select * from first
    -> inner join second
    -> ON first.ID = second.ID;
+------+--------+------+--------+
| ID   | name   | ID   | name   |
+------+--------+------+--------+
|    3 | raman  |    3 | raman  |
|    2 | chaman |    2 | chaman |
+------+--------+------+--------+
2 rows in set (0.00 sec)

mysql>
```

```
mysql> use pratical
Database changed
mysql> select * from first
    -> LEFT Join second
    -> ON first.ID = Second.ID;
+------+--------+------+--------+
| ID   | name   | ID   | name   |
+------+--------+------+--------+
|    1 | aman   | NULL | NULL   |
|    2 | chaman |    2 | chaman |
|    3 | raman  |    3 | raman  |
|    4 | mayur  | NULL | NULL   |
+------+--------+------+--------+
4 rows in set (0.00 sec)

mysql>
```

```
mysql> select * from first
    -> RIGHT Join second
    -> ON first.ID = Second.ID;
+------+--------+------+-----------+
| ID   | name   | ID   | name      |
+------+--------+------+-----------+
|    3 | raman  |    3 | raman     |
|    2 | chaman |    2 | chaman    |
| NULL | NULL   |    5 | bhupendra |
| NULL | NULL   |    6 | kartik    |
+------+--------+------+-----------+
4 rows in set (0.00 sec)

mysql>
```

Experiment 7

Query optimization through Nested Query (Use of logical connectives, set comparison operators, Union, Intersect,

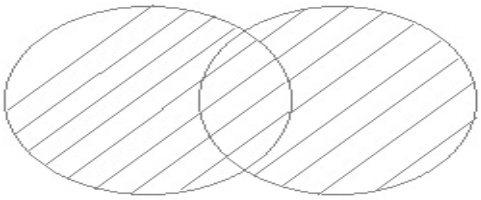
Except, Exists clauses)

- **Run Basic queries involving nested/subqueries**
- **Use of in/not in operator for nested queries**
- **Use of all, some, exists, not exists for nested queries**

Union: Returns all distinct rows selected by both the queries

Syntax:

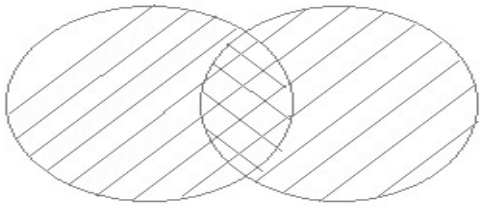
Query1 Union Query2;



Union all: Returns all rows selected by either query including the duplicates.

Syntax:

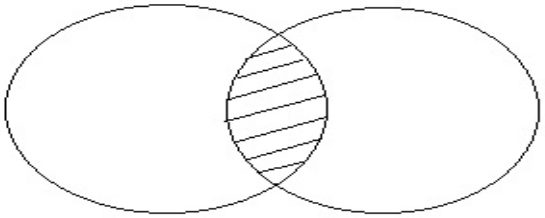
Query1 Union all Query2;



Intersect: Returns rows selected that are common to both queries.

Syntax:

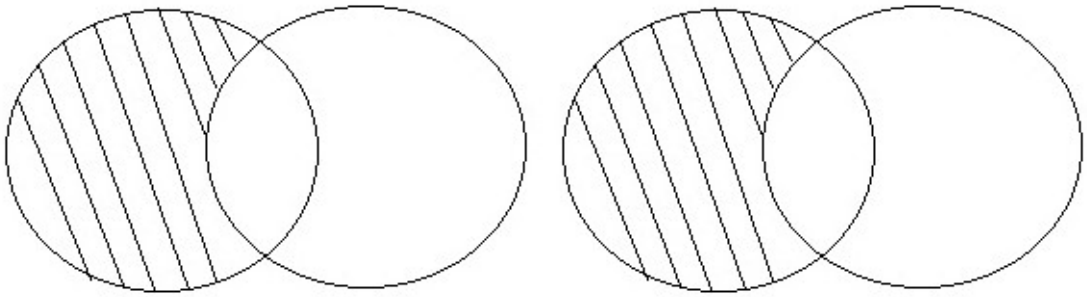
Query1 Intersect Query2;

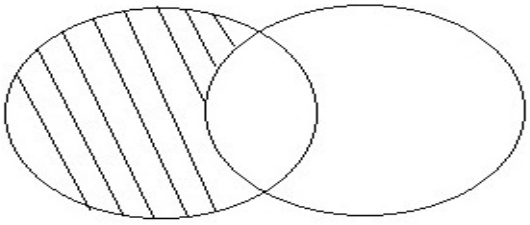


Minus: Returns all distinct rows selected by the first query and are not by the second

Syntax:

Query1 minus Query2;





First Table

ID	NAME
2	adam
3	chester

Second Table

ID	NAME
1	abhi
2	adam

a)Select *from First UNION select * from Second

ID	NAME
1	abhi
2	adam
3	chester

Ans

b) Select *from First UNION ALL select * from Second

ID	NAME
1	abhi
2	adam
2	adam
3	chester

Ans

c) Select *from First INTERSECT select * from Second

ID	NAME
2	adam

d) Select *from First MINUS select * from Second

ID	NAME
1	abhi

d) Queries:

Q1: Display all the dept numbers available with the dept and emp tables avoiding duplicates.

Solution:

1. Use select from clause. 2. Use union select clause to get the result.

Ans:

SQL> select deptno from emp union select deptno from dept;

DEPTNO

1

2

12

30

40

Q2: **Display all the dept numbers available with the dept and emp tables.**

Solution:

1. Use select from clause. 2. Use union all in select clause to get the result. Ans:

SQL> select deptno from emp union all select deptno from dept;

DEPTNO

1

2
2

1
12

1
2

30
40

9 rows selected.

Q3: **Display all the dept numbers available in emp and not in dept tables and vice versa.**

Solution:

- Use select from clause.

- Use minus in select clause to get the result. Ans:

SQL> select deptno from emp minus select deptno from dept;

DEPTNO

12

SQL> select deptno from dept minus select deptno from emp; DEPTNO

30
40

SQL QUERY:

```
mysql> select * from first
    -> UNION
    -> select * from second;
+------+-----------+
| ID   | name      |
+------+-----------+
|    1 | aman      |
|    2 | chaman    |
|    3 | raman     |
|    4 | mayur     |
|    5 | bhupendra |
|    6 | kartik    |
+------+-----------+
6 rows in set (0.00 sec)

mysql>
```

```
mysql> select * from first
    -> UNION ALL
    -> select * from second;
+------+-----------+
| ID   | name      |
+------+-----------+
|    1 | aman      |
|    2 | chaman    |
|    3 | raman     |
|    4 | mayur     |
|    3 | raman     |
|    2 | chaman    |
|    5 | bhupendra |
|    6 | kartik    |
+------+-----------+
8 rows in set (0.00 sec)

mysql>
```

```
mysql> select * from first
    -> INTERSECT
    -> select * from second;
+------+--------+
| ID   | name   |
+------+--------+
|    2 | chaman |
|    3 | raman  |
+------+--------+
2 rows in set (0.00 sec)

mysql>
```

```
mysql> select * from first
    -> EXCEPT
    -> select * from second;
+------+-------+
| ID   | name  |
+------+-------+
|    1 | aman  |
|    4 | mayur |
+------+-------+
2 rows in set (0.00 sec)

mysql>
```

Experiment 8:

Use of Group By and Having clause,

- Learn about group by operator and run queries using group by operator
- Run queries using group by and having operators
- Use of aggregate operators with group by and having Operators

The MySQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

The following SQL statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

The following SQL statement lists the number of customers in each country, sorted high to low:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

MySQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

lists the number of customers in each country. Only include countries with more than 5 customers:

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

SQL QUERY:

```
| jhon     | Analyst      | 2020-05-03   |           10 |
| Harshad  | Scientist    | 2024-06-30   |           10 |
+----------+--------------+--------------+--------------+
6 rows in set (0.00 sec)

mysql> select name,occupation from emp
    -> group by working_hour;
```

```
mysql> SELECT name
    -> FROM emp
    -> WHERE working_hour = 10
    -> GROUP BY name, occupation;
+---------+
| name    |
+---------+
| Amit    |
| jhon    |
| Harshad |
+---------+
3 rows in set (0.00 sec)
```

Experiment 9:

Index creation through SQL

- **Learn to create index in MYSQL**
- **View index**
- **Change ordering of index key**
- **Compare time taken to search with and without using Index**

MySQL CREATE INDEX

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index_name

ON table_name (column1, column2, ...);

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

if you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

The DROP INDEX statement is used to delete an index in a table.

```
ALTER TABLE table_name
DROP INDEX index_name;
```

SQL QUERY:

```
mysql> CREATE INDEX idx_Lastname
    -> ON emp(name);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> CREATE INDEX idx_pname
    -> ON emp(name,occupation);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> alter table emp
    -> drop index idx_pname;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

PRACTICAL NO 10

Aim: To Study the Concept of Trigger in DBMS

Theory:

Triggers

Triggers are the SQL statements that are automatically executed when there is any change in the database. The triggers are executed in response to certain events (INSERT, UPDATE or DELETE) in a particular table. These triggers help in maintaining the integrity of the data by changing the data of the database in a systematic fashion.

Syntax

create trigger Trigger_name

(before | after)

[insert | update | delete]

on [table_name]

[for each row]

[trigger_body]

1. CREATE TRIGGER: These two keywords specify that a triggered block is going to be declared.
2. TRIGGER_NAME: It creates or replaces an existing trigger with the Trigger_name. The trigger name should be unique.
3. BEFORE | AFTER: It specifies when the trigger will be initiated i.e. before the ongoing event or after the ongoing event.
4. INSERT | UPDATE | DELETE: These are the DML operations and we can use either of them in a given trigger.
5. ON [TABLE_NAME]: It specifies the name of the table on which the trigger is going to be applied.
6. FOR EACH ROW: Row-level trigger gets executed when any row value of any column changes.
7. TRIGGER BODY: It consists of queries that need to be executed when the trigger is called.

Example

Suppose we have a table named Student containing the attributes Student_id, Name, Address, and Marks.

Now, we want to create a trigger that will add 100 marks to each new row of the Marks column whenever a new student is inserted to the table.

The SQL Trigger will be:

```
CREATE TRIGGER Add_marks
BEFORE
INSERT
ON Student
FOR EACH ROW
SET new.Marks = new.Marks + 100;
```

The new keyword refers to the row that is getting affected.

After creating the trigger, we will write the query for inserting a new student in the database.

```
INSERT INTO Student (Name, Address, Marks) VALUES (&#39;Alizeh&#39;, &#39;Maldives&#39;, 110);
```

The Student_id column is an auto-increment field and will be generated automatically when a new record is inserted into the table.

To see the final output the query would be:

```
SELECT * FROM Student;
```

Advantages of Triggers

1. Triggers provide a way to check the integrity of the data. When there is a change in the database the triggers can adjust the entire database.
2. Triggers help in keeping User Interface lightweight. Instead of putting the same function call all over the application you can put a trigger and it will be executed.

Disadvantages of Triggers

1. Triggers may be difficult to troubleshoot as they execute automatically in the database. If there is some error then it is hard to find the logic of trigger because they are fired before or after updates/inserts happen.
2. The triggers may increase the overhead of the database as they are executed every time any field is updated.

SQL QUERY:

```
mysql> CREATE TABLE t_school (
    -> ID int,
    -> School_name varchar(40),
    -> no_of_students int,
    -> not_of_teachers int,
    -> email_id varchar(40));
Query OK, 0 rows affected (0.06 sec)

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

mysql> insert into students values (
    -> 1, "Billie", "New York", 220),
    -> (2, "Eilish", "London", 190),
    -> (3, "Ariana", "Miami", 180);
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE TRIGGER Add_marks
    -> BEFORE
    -> INSERT
    -> ON students
    -> FOR EACH ROW
    -> SET new.Marks=new.Marks
    -> +100;
Query OK, 0 rows affected (0.02 sec)

mysql> Select * from students;
+------------+--------+----------+-------+
| Student_id | Name   | Address  | Marks |
+------------+--------+----------+-------+
|          1 | Billie | New York |   220 |
|          2 | Eilish | London   |   190 |
|          3 | Ariana | Miami    |   180 |
+------------+--------+----------+-------+
3 rows in set (0.01 sec)
```

PRACTICAL NO 11

Aim: To Study the Transaction Control Language

Theory: Transaction control language (TCL).

· TCL stands for Transaction Control Language.

· This command is used to manage the changes made by DML statements.

· TCL allows the statements to be grouped together into logical transactions.

TCL commands are as follows:

1.COMMIT

2.SAVEPOINT

3.ROLLBACK

4. SET TRANSACTION

Suppose we have inserted some records in to Employee table. Now we need to save them.

How? Similarly, we have updated something wrong on the table. After updating we realized that its wrong.

Now we need to unsave the changes that have been done.

How? We have deleted something or inserted something which is not correct. It has to be undone. All these saving and undoing the tasks can be done by TCL.

COMMIT

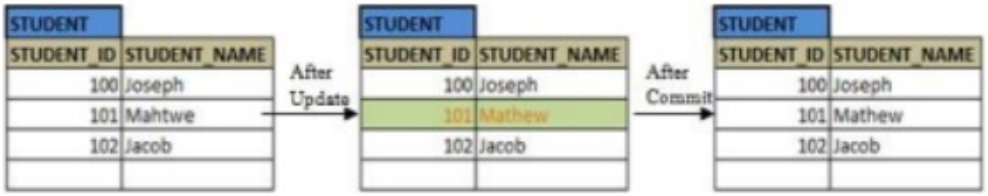
COMMIT saves the transaction on the database. The transaction can be insert, delete or update. Once the COMMIT is issued, the changes are savedpermanently in the database. It cannot be undone.

UPDATE STUDENT SET STUDENT_NAME = ‘Mathew’ WHERE SRUDENT_NAME = ‘Mahtwe’;

COMMIT;

Above set of transactions, updates the wrong student name to the correct one and saves the changes permanently in the database. Update transactionis complete only when commit is issued, else there will be lock on ‘Mahtwe’ record till the commit or rollback is issued.

Below diagram shows that ‘Mahtwe’ is updated to ‘Mathew’ and still therewill be a lock on his record. Once Commit is issued, updated value is permanently saved to database and lock is released.

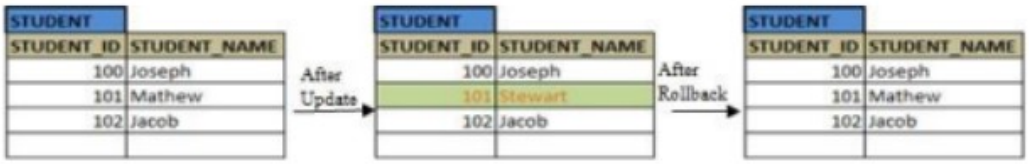


ROLLBACK

ROLLBACK command is used to undo the insert, delete or update transactionin the database. It undoes the transaction performed on the table and restores the previous stored value.

UPDATE STUDENT SET STUDENT_NAME = ‘Stewart’ WHERE STUDENT_NAME = ‘Mathew;

ROLLBACK;



Here, after updating the student name, user realizes that he has updated the wrong record and he wants to undo his update. What he does is, he issues ROLLBACK command and undoes his update. When he issues update statement Mathew’s record will be locked for update and will be updated to ‘Stewart’.But lock will not be released – meaning update is not saved fully into the database and transaction is not complete. Once the rollback is issued, it undoes the update and restores the value to ‘Mathew’ and save thechanges permanently. Hence there will not be any changes done to Mathew.

SAVEPOINT

Suppose there are set of update, delete transactions performed on the tables. But there are some transactions which we are very sure about correctness.

After that set of transactions we are uncertain about the correctness. So what we can do here is we can set a SAVEPOINT at the correct transaction telling the database that, in case of rollback, rollback till the savepoint marked. Hence the changes done till savepoint will be unchanged and all the transactions after that will be rolled back.

Have look at below transactions.

o It updates ‘Mahtwe’ to ‘Mathew’. Hence we have lock on Mathew record.

o It also updates Joseph’s record for his Age to 15.

Here, say we have set the SAVEPOINT after first transaction. Then second transaction for age update is issued and we see that it is a wrong update, but the name update is correct.

Here we have to rollback only the last transaction and retain the first transaction. So we issue Rollback till the savepoint.

What it does is, it reverts all the transaction till the savepoint. Although there is no commit, the transactions till savepoint is retained and saved later upon commit. Hence you can see the lock on Mathew’s record.

UPDATE STUDENT SET STUDENT_NAME = ‘Mathew’ WHERE STUDENT_NAME = ‘Mahtwe’;

SAVEPOINT S1;

UPDATE STUDENT SET AGE = 15 WHERE STUDENT_ID = 100;

ROLLBACK to S1;

In the case of multiple transactions, savepoint can be given after each transactions and transaction can be rolled back to any of the transactions.

TRANSACTION T1;

— Transaction can be insert, update or delete

SAVEPOINT S1;

TRANSACTION T2;

SAVEPOINT S2;

TRANSACTION T3;

SAVEPOINT S3;

TRANSACTION T4;

ROLLBACK TO S1; -- This will rollback all the changes by T1 andT2

and will have only the changes done on T1.

AUTOCOMMIT

AUTOCOMMIT command automatically commits each transaction after its execution. If this command is set, then no need to explicitly issue commit.

We cannot rollback our transactions, if AUTOCOMMIT is on. This needs to beset /unset before we begin any transactions.

SET AUTOCOMMIT ON; -- Sets AUTOCOMMIT to ON

SET AUTOCOMMIT ON; -- Sets AUTOCOMMIT to OFF

SQL QUERY:

```
mysql> use school;
Database changed
mysql> insert into t_school
    -> values (1, "Boys Town Public School", 100, 80, "btps@gmail.com"),
    -> (2, "Guru Govind Singh Public School", 800, 35, "ggps@gmail.com"),
    -> (3, "Delhi Public School", 1200, 30, "dps@gmail.com"),
    -> (4, "Ashoka Universal School", 1100, 40, "aus@gmail.com"),
    -> (5, "Calibers English Medium School", 9000, 31,"cems@gmail.com");
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from t_school;
+------+---------------------------------+---------------+-----------------+----------------+
| ID   | School_name                     | no_of_students | not_of_teachers | email_id       |
+------+---------------------------------+---------------+-----------------+----------------+
|    1 | Boys Town Public School         |           100 |              80 | btps@gmail.com |
|    2 | Guru Govind Singh Public School |           800 |              35 | ggps@gmail.com |
|    3 | Delhi Public School             |          1200 |              30 | dps@gmail.com  |
|    4 | Ashoka Universal School         |          1100 |              40 | aus@gmail.com  |
|    5 | Calibers English Medium School  |          9000 |              31 | cems@gmail.com |
+------+---------------------------------+---------------+-----------------+----------------+
5 rows in set (0.00 sec)
```

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

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

mysql> Start transaction;
Query OK, 0 rows affected (0.00 sec)

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

mysql> update t_school Set no_of_students = 9050 where ID=5;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> Select * from t_school;
+------+---------------------------------+----------------+-----------------+----------------+
| ID   | School_name                     | no_of_students | not_of_teachers | email_id       |
+------+---------------------------------+----------------+-----------------+----------------+
|    1 | Boys Town Public School         |            100 |              80 | btps@gmail.com |
|    2 | Guru Govind Singh Public School |            800 |              35 | ggps@gmail.com |
|    3 | Delhi Public School             |           1200 |              30 | dps@gmail.com  |
|    4 | Ashoka Universal School         |           1100 |              40 | aus@gmail.com  |
|    5 | Calibers English Medium School  |           9050 |              31 | cems@gmail.com |
+------+---------------------------------+----------------+-----------------+----------------+
5 rows in set (0.00 sec)
```

```
mysql> Rollback to insertion;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from t_school;
+------+---------------------------------+----------------+-----------------+----------------+
| ID   | School_name                     | no_of_students | not_of_teachers | email_id       |
+------+---------------------------------+----------------+-----------------+----------------+
|    1 | Boys Town Public School         |            100 |              80 | btps@gmail.com |
|    2 | Guru Govind Singh Public School |            800 |              35 | ggps@gmail.com |
|    3 | Delhi Public School             |           1200 |              30 | dps@gmail.com  |
|    4 | Ashoka Universal School         |           1100 |              40 | aus@gmail.com  |
|    5 | Calibers English Medium School  |           9000 |              31 | cems@gmail.com |
+------+---------------------------------+----------------+-----------------+----------------+
5 rows in set (0.00 sec)
```