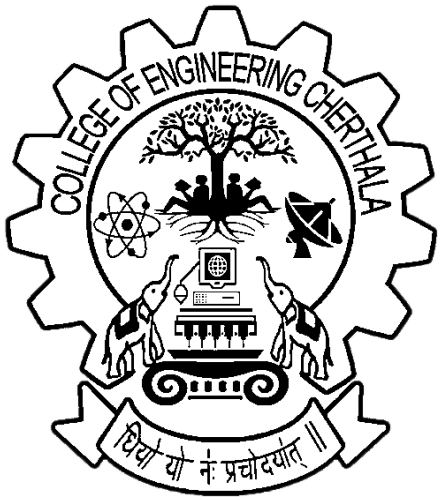
COLLEGE OF ENGINEERING CHERTHALA

LAB RECORD

20MCA134 – ADVANCED DBMS LAB



CERTIFICATE

This is certified to be bona fide works of Mr./Ms. ……., In the class ..... , Reg. No. ……., of College of Engineering Cherthala, during the academic year 2024-25.

Teacher In Charge **External Examiner**

Internal Examiner **Date:**

Experiment no: 1

MySQL installation

Install and configure client and server for MySQL (Show all commands and necessary steps for installation and configuration).

MYSQL:

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons −

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

MySQL Features ○ **Relational Database Management System (RDBMS):** MySQL is a relational database management system.

- **Easy to use:** MySQL is easy to use. You have to get only the basic knowledge of SQL. You can build and interact with MySQL with only a few simple SQL statements.
- **It is secure:** MySQL consist of a solid data security layer that protects sensitive data from intruders. Passwords are encrypted in MySQL.
- **Client/ Server Architecture:** MySQL follows a client /server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they query data, save changes, etc.
- **Free to download:** MySQL is free to use and you can download it from MySQL official website.
- **It is scalable:** MySQL can handle almost any amount of data, up to as much as 50 million rows or more. The default file size limit is about 4 GB. However, you can increase this number to a theoretical limit of 8 TB of data.
- **Compatibale on many operating systems:** MySQL is compatible to run on many operating systems, like Novell NetWare, Windows* Linux*, many varieties of UNIX*

also provides a facility that the clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).

- **Allows roll-back:** MySQL allows transactions to be rolled back, commit and crash recovery.
- **High Performance:** MySQL is faster, more reliable and cheaper because of its unique storage engine architecture. ○ **High Flexibility:** MySQL supports a large number of embedded applications which makes MySQL very flexible.
- **High Productivity:** MySQL uses Triggers, Stored procedures and views which allows the developer to give a higher productivity.

Installation Step of MYSQL:

Following steps are to be followed while installing MYSQL on Ubuntu operating System.

1. Connect the system with the Internet.
2. Open the terminal and Execute the command sudo apt-get update
   sudo apt-get install mysql-server
3. Enter the password for root as **"root"**
4. After installation enter the below command to get the MYSQL Terminal. mysql –u root -p
5. Enter the earlier chosen password ie. **root**
6. Then enter the command "show database", by this all the databases in the system will display on screen.
7. Create database.
8. Use that given database.

Result: The installation of MySQL is done successfully.

Experiment no: 2

DCL and DDL commands in SQL

Aim : Design any database with at least 3 entities and relationships between them. Apply DCL and DDL commands.

Objective:

- To understand the different issues involved in the design and implementation of a database system
- To understand and use Data Definition Language and Data Control Language to write query for a database ***Theory:***

DATA DEFINITION LANGUAGE (DDL): The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. **Some commands of DDL are:**

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

1. CREATE:

(a) CREATE DATABASE: You can create a MySQL database by using MySQL Command

Syntax:

CREATE DATABASE database_name;

Example:

Let's take an example to create a database name "employees"

CREATE DATABASE employees;

We can check the created database by the following query:

SHOW DATABASES;

(b) USE DATABASE: Used to select a particular database.

Syntax:

USE database_name;

Example: Let's take an example to use a database name "customers".

USE customers;

(c) DROP DATABASE: You can drop/delete/remove a MySQL database easily with the MySQL command. You should be careful while deleting any database because you will lose your all the data available in your database.

Syntax:

DROP DATABASE database_name;

Example: Let's take an example to drop a database name "employees"

DROP DATABASE employees;

(d) CREATE TABLE: This is used to create a new relation (table)

The MySQL CREATE TABLE command is used to create a new table into the database.

Syntax:

Following is a generic syntax for creating a MySQL table in the database.

CREATE TABLE table_name (column_name column_type...);

Example:

Here, we will create a table named "student" in the database "mydatabase".

CREATE TABLE cus_tbl (roll_no INT NOT NULL, fname VARCHAR(100) NOTNULL,

surname VARCHAR(100) NOT NULL, PRIMARY KEY (roll_no));

See the created table: Use the following command to see the table already created:

SHOW tables;

See the table structure: Use the following command to see the table already created:

DESCRIBE table_name;

2. ALTER:

MySQL ALTER statement is used when you want to change the name of your table or any table field. It is also used to add or delete an existing column in a table.

The ALTER statement is always used with "ADD", "DROP" and "MODIFY" commands according to the situation.

(a) ALTER TABLE ...ADD...: This is used to add some extra fields into existing relation.

Syntax: ALTER TABLE relation_name ADD (newfield_1 data_type(size), newfield_2 data_type(size),..);

Example: ALTER TABLE student ADD (Address CHAR(10));

(b) ALTER TABLE...MODIFY...: This is used to change the width as well as data type of fields of existing relations.

Syntax: ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2 newdata_type(Size),. .. field_newdata_type(Size));

Example: ALTER TABLE student MODIFY(fname VARCHAR(10),class VARCHAR(5));

(c) ALTER TABLE..DROP ..... This is used to remove any field of existing relations.

Syntax: ALTER TABLE relation_name DROP COLUMN (field_name);

Example: ALTER TABLE student DROP column (sname);

(d) ALTER TABLE..RENAME...: This is used to change the name of fields in existing relations.

Syntax: ALTER TABLE relation_name RENAME COLUMN (OLD field_name) to (NEW field_name);

Example: ALTER TABLE student RENAME COLUMN sname to stu_name;

3. RENAME: It is used to modify the name of the existing database object.

Syntax: RENAME TABLE old_relation_name TO new_relation_name;

Example: RENAME TABLE studentd TO studentd1;

4. TRUNCATE and DROP

Difference between Truncate & Drop:-

RUNCATE: This command will remove the data permanently. But structure will not be removed. DROP: This command will delete the table data and structure permanently.

Syntax: TRUNCATE TABLE <Table name>

Example TRUNCATE TABLE student;

Syntax: DROP TABLE <Table name>

Example DROP TABLE student;

Data Control Language(DCL) : Thisis used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges.

DCL defines two commands,

- **Grant:** Gives user access privileges to database.
- **Revoke:** Take back permissions from user.

Syntax: GRANT privilege_name ON object_name TO user_name;

Example: GRANT CREATE TABLE TO user1;

Syntax: REVOKE privilege_name ON object_name FROM user_name;

Example: REVOKE CREATE TABLE FROM user1;

LAB PRACTICE ASSIGNMENT:

Consider the following table structures for this assignment:

Table Name 1: **CUSTOMER**

Fields:

Cust_idvarchar(10) Primary Key,C_nameVarchar(15) Not NULL,City varchar(10).

Table Name 2: **BRANCH**

Fields:

Branch_idVarchar(5) Primary Key, bnameVarchar (15), City varchar(10).

Table Name 3: **DEPOSIT**

Fields:

Acc_novarchar(10) Primary Key, Cust_id varchar(10) Not NULL, Amount int, Branch_id varchar(5), Open_date date.

Table Name 4: **BORROW**

Fields:

Loan_no varchar(5) Primary Key, Cust_id varchar (10), Branch_id varchar(5), Amount int.

Perform the following command/operation on the above table:

1) Create a Database
2) Show Database
3) Use Database
4) Drop Database
5) Create tables and Describe that Tables
6) Alter Command
   i) Add column address to Customer table
   ii) Modify any column
   iii) Rename column address to new_address
   iv) Drop column address from Customer table
   v) Rename table Branch to Branch1
6) Perform DCL Commands Grant and Revoke on Customer table
7) Truncate table
8) Drop table

OUTPUT

1) Create a Database

```
mysql> create database bank;
Query OK, 1 row affected (0.15 sec)
```

2) Show Database

```
mysql> show databases;
+--------------------------+
| Database                 |
+--------------------------+
| abhirami                 |
| bank                     |
| information_schema       |
| mysql                    |
| n                        |
| performance_schema       |
| sys                      |
| vishnu                   |
+--------------------------+
8 rows in set (0.05 sec)
```

3) Use Database

```
mysql> use bank;
Database changed
```

4) Drop Database

```
mysql> drop database abhirami;
Query OK, 1 row affected (0.49 sec)
```

5) Create tables and Describe that Tables

```
mysql> create table customer(cust_id varchar(10)primary key,c_name
varchar(15)NOT NULL,city varchar(10));
Query OK, 0 rows affected (0.89 sec)
```

```
mysql> create table branch(branch_id varchar(5)primary key,bname
varchar(15),city varchar(10));
Query OK, 0 rows affected (0.55 sec)
```

```
mysql> create table deposit(acc_no varchar(10)primary key,cust_id
varchar(10)NOT NULL,amt int,branch_id varchar(5),open_date date);
Query OK, 0 rows affected (0.41 sec)
```

```
mysql> create table borrow(loan_no varchar(5)primary key,cust_id
varchar(10),branch_id varchar(5),amount int);
Query OK, 0 rows affected (0.36 sec)
```

```
mysql> desc customer;
+------------+-----------------+--------+-------+------------+----------+
| Field    | Type          | Null | Key | Default | Extra |
+------------+-----------------+--------+-------+------------+----------+
| cust_id | varchar(10) | NO    | PRI | NULL     |        |
| c_name   | varchar(15) | NO    |      | NULL     |        |
| city     | varchar(10) | YES  |      | NULL     |        |
+------------+-----------------+--------+-------+------------+----------+
3 rows in set (0.02 sec)
```

6) Alter Command

6.i. Add column address to Customer table

```
mysql> alter table customer add column address varchar(10);
Query OK, 0 rows affected (0.26 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> desc customer;
+------------+-----------------+--------+-------+------------+----------+
| Field    | Type          | Null | Key | Default | Extra |
+------------+-----------------+--------+-------+------------+----------+
| cust_id | varchar(10) | NO    | PRI | NULL     |        |
| c_name   | varchar(15) | NO    |      | NULL     |        |
| city     | varchar(10) | YES  |      | NULL     |        |
| address | varchar(10) | YES  |      | NULL     |        |
+------------+-----------------+--------+-------+------------+----------+
4 rows in set (0.02 sec)
```

6.ii. Modify any column

```
mysql> alter table customer modify column cust_id varchar(20);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> desc customer;
+------------+-----------------+--------+-------+------------+----------+
| Field    | Type          | Null | Key | Default | Extra |
+------------+-----------------+--------+-------+------------+----------+
| cust_id | varchar(20) | NO    | PRI | NULL     |        |
| c_name   | varchar(15) | NO    |      | NULL     |        |
| city     | varchar(10) | YES  |      | NULL     |        |
| address | varchar(10) | YES  |      | NULL     |        |
+------------+-----------------+--------+-------+------------+----------+
4 rows in set (0.02 sec)
```

6.iii. Rename column address to new_address

```
mysql> alter table customer rename column address to new_address;
Query OK, 0 rows affected (0.21 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> desc customer;
+-----------------+-----------------+--------+-------+------------+----------+
| Field           | Type            | Null   | Key   | Default    | Extra    |
+-----------------+-----------------+--------+-------+------------+----------+
| cust_id         | varchar(20)     | NO     | PRI   | NULL       |          |
| c_name          | varchar(15)     | NO     |       | NULL       |          |
| city            | varchar(10)     | YES    |       | NULL       |          |
| new_address     | varchar(10)     | YES    |       | NULL       |          |
+-----------------+-----------------+--------+-------+------------+----------+
4 rows in set (0.02 sec)
```

6.iv. Drop column address from Customer table

```
mysql> alter table customer drop column new_address;
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc customer;
+------------+-----------------+--------+-------+------------+----------+
| Field      | Type            | Null   | Key   | Default    | Extra    |
+------------+-----------------+--------+-------+------------+----------+
| cust_id    | varchar(20)     | NO     | PRI   | NULL       |          |
| c_name     | varchar(15)     | NO     |       | NULL       |          |
| city       | varchar(10)     | YES    |       | NULL       |          |
+------------+-----------------+--------+-------+------------+----------+
3 rows in set (0.02 sec)
```

6.v. Rename table Branch to Branch1

```
mysql> alter table branch rename to branch1;
Query OK, 0 rows affected (0.44 sec)

mysql> show tables;
+---------------------+
| Tables_in_bank      |
+---------------------+
| borrow              |
| branch1             |
| customer            |
| deposit             |
+---------------------+
 rows in set (0.00 sec)
```

7) Perform DCL Commands Grant and Revoke on Customer table

7.i. Grant : Gives user access privileges to database.

```
mysql> create user user1 identified by 'abc123';
Query OK, 0 rows affected (0.16 sec)
```

```
mysql> grant all on customer to user1;
Query OK, 0 rows affected (0.09 sec)
mysql> exit
Bye

cec@user:~$ sudo mysql -u user1 -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.36-0ubuntu0.20.04.1 (Ubuntu)
Copyright (c) 2000, 2024, Oracle and/or its affiliates
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.

mysql> show databases;
+--------------------------+
| Database                 |
+--------------------------+
| bank                     |
| information_schema       |
| performance_schema       |
+--------------------------+
3 rows in set (0.00 sec)
```

- ```
  mysql> use bank;
  Reading table information for completion of table and column names You
  can turn off this feature to get a quicker startup with -A
  Database changed
  ```

- ```
  mysql> show tables;
  +----------------------+
  | Tables_in_bank |
  +----------------------+
  | customer       |
  +----------------------+
  1 row in set (0.00 sec)
  ```

- ```
  mysql> insert into customer values('a1','binu','chennai');
  Query OK, 1 row affected (0.11 sec)
  ```

- ```
  mysql> insert into customer values('a2','jinu','alpy');
  Query OK, 1 row affected (0.14 sec)
  ```

- ```
  mysql> insert into customer values('a4','diya','tvm');
  Query OK, 1 row affected (0.08 sec)
  ```
  ```

- mysql> select*from customer;

```
+-----------+----------+-----------+-----------------+
| cust_id | c_name | city     | new_address |
+-----------+----------+-----------+-----------------+
| a1       | binu    | chennai | NULL         |
| a2       | jinu    | alpy     | NULL         |
| a4       | diya    | tvm      | NULL         |
+-----------+----------+-----------+-----------------+
3 rows in set (0.00 sec)
```

- mysql> exit
  Bye

7.ii. Revoke : Take back permissions from user.

- cec@user:~$ sudo mysql -u root
  Welcome to the MySQL monitor. Commands end with ; or \g.
  Your MySQL connection id is 15
  Server version: 8.0.36-0ubuntu0.20.04.1 (Ubuntu)

- mysql> use bank;
  Reading table information for completion of table and column names
  You can turn off this feature to get a quicker startup with -A
  Database changed

- mysql> revoke all on customer from user1;
  Query OK, 0 rows affected (0.08 sec)
  mysql> exit;
  Bye

- cec@user:~$ sudo mysql -u user1 -p;
  Enter password:
  Welcome to the MySQL monitor. Commands end with ; or \g.
  Your MySQL connection id is 17
  Server version: 8.0.36-0ubuntu0.20.04.1 (Ubuntu)

- mysql> show databases;

```
+--------------------------+
| Database           |
+--------------------------+
| information_schema |
| performance_schema |
+--------------------------+
2 rows in set (0.00 sec)
```

- mysql> exit;
  Bye

8) Truncate table

- ```
  mysql> truncate table customer;
  Query OK, 0 rows affected (0.92 sec)
  ```

- ```
  mysql> select*from customer;
  Empty set (0.01 sec)
  ```

9) Drop table

- ```
  mysql> Drop table customer;
  Query OK, 0 rows affected (0.92 sec)
  ```

Result: The program is executed successfully and the output is obtained
  ```

Experiment no: 3

DML commands in SQL

Aim :

Design and implement a database and apply at least 10 different DML queries for the following task. For a given input string display only those records which match the given pattern or a phrase in the search string. Make use of wild characters and LIKE operator for the same. Make use of Boolean and arithmetic operators wherever necessary.

Objective :

- To understand the different issues involved in the design and implementation of a database system
- To understand and use Data Manipulation Language to query to manage a database

Theory :

DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

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

1. **INSERT INTO:** This is used to add records into a relation. These are three type of INSERT INTO queries which are as:

a) Inserting a single record

Syntax: INSERT INTO < relation/table name> **(field_1,field_2……**field_n)VALUES (data_1,data_2, ...data_n);

Example: INSERT INTO student(sno, sname, address)VALUES(1,'Ravi','M.Tech','Palakol');

b) To insert multiple record

Here, we are going to insert record in the "cus_tbl" table of "customers" database.

Syntax: INSERT INTO table_name (column1, column2, ..., columnN)
VALUES
(value1_row1, value2_row1, ..., valueN_row1),
(value1_row2, value2_row2, ..., valueN_row2),
...,
(value1_rowN, value2_rowN, ..., valueN_rowN);

Example: INSERT INTO cus_tbl(cus_id, cus_firstname, cus_surname) VALUES
(5, 'Ajeet', 'Maurya'),
(6, 'Deepika', 'Chopra'),
(7, 'Vimal', 'Jaiswal');

2. **SELECT:** This is used to Retrieve data from one or more tables.

a) SELECT FROM: To display all fields for all records.

Syntax : SELECT * FROM relation_name;

Example : SQL> select * from dept;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

b) SELECT - FROM -WHERE: This query is used to display a selected set of fields for a selected set of records of a relation.

Syntax: SELECT a set of fields FROM relation_name WHERE condition;

Example: SQL> select * FROM dept WHERE deptno<=20;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS

c) SELECT - FROM -WHERE- LIKE

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

Syntax: SELECT *column1, column2,...*FROM *table_name* WHERE *column* LIKE *pattern*;

Example: SELECT * FROM Customers WHERE CustomerName LIKE 'a%';

LIKE Operator	Description
WHERE CustomerName LIKE ‘a%’	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

d) SELECT – DISTINCT

The SELECT DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values. The SELECT DISTINCT statement is used to return only distinct (different) values.

Syntax: SELECT DISTINCT *column1, column2, . . .* FROM *table_name*;

Example: SELECT COUNT(DISTINCT Country) FROM Customers;

e) SELECT - BETWEEN

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

Syntax: SELECT *column_name(s)* FROM *table_name* WHERE *column_name* BETWEEN *value1* AND *value2;*

Example: SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;

f) WHERE with - *AND* LOGICAL Operator

The WHERE clause when used together with the AND logical operator, is only executed if ALL filter criteria specified are met.

```
SELECT * FROM `movies` WHERE `category_id` = 2 AND `year_released` = 2008;
```

g) WHERE with - *OR* LOGICAL Operator

The WHERE clause when used together with the OR operator, is only executed if any or the entire specified filter criteria is met.

The following script gets all the movies in either category 1 or category 2

```
SELECT * FROM `movies` WHERE `category_id` = 1 OR `category_id` = 2;
```

h) **WHERE with - *Arithmetic* Operator**

Operator	Description	Example
=	Checks if the values of the two operands are equal or not, if yes, then the condition becomes true.	(A = B) is not true.
!=	Checks if the values of the two operands are equal or not, if the values are not equal then the condition becomes true.	(A != B) is true.

>	Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of the left operand is less than the value of the right operand, if yes then the condition becomes true.	(A < B) is true.
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.	(A >= B) Is not true.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.	(A <= B) is true.

Example: SELECT agent_code, agent_name, working_area,(commission * 2) AS double_commission FROM agents WHERE (commission * 2) > 0.25;

3. **UPDATE-SET-WHERE:** This is used to update the content of a record in a relation.

Syntax: UPDATE relation name SET Field_name1=data,field_name2=data, WHERE field_name=data;

Example: UPDATE student SET **sname = 'kumar' WHERE sno=1;**

4. **DELETE-FROM**: This is used to delete all the records of a relation but it will retain the structure of that relation.

a) **DELETE-FROM**: This is used to delete all the records of relation.

Syntax: DELETE FROM relation_name; ***Example:***

Example: DELETE FROM std;

b) **DELETE -FROM-WHERE:** This is used to delete a selected record from a relation.

Syntax: DELETE FROM relation_name WHERE condition;

Example: DELETE FROM student WHERE sno = 2;

LAB PRACTICE ASSIGNMENT:

Consider the following table structure for this assignment:

CUSTOMER(Cust_id, C_name, City)
BRANCH(Branch_id, bname, City)
DEPOSIT(Acc_no , Cust_id, Amount, Branch_id, Open_date)
BORROW(Loan_no, Cust_id, Branch_id, Amount)

Perform the following queries on the above table:

1) Insert minimum 10 rows on each table and display that data.
2) List Cust_id along with customer name.
3) List Cust_id of customers having amount greater than 10000.
4) List account date of customer 'sulu'.
5) List Cust_id of customers who have opened account after 01/01/2016.
6) List account no. and Cust_id of customers having amount between 40,000 and 80,000.
7) List customer name starting with 'S'.
8) List customer from depositor starting with '_a%'.
9) List customer name, account no and amount from deposit having exactly 5 characters in name.
10) List Cust_id, Loan no and Loan amount of borrowers.
11) List cust_id and C_name of depositors.
12) List all the customers who are depositors but not borrowers.
13) List all the customers who are both depositors and borrowers.
14) List all the customers along with their amount who are either borrowers.
15) List the cites of depositor having branch 'Cherthala'.
16) Update 10% interest to all depositors.
17) Update 10% to all depositors living in 'thrissur'.
18) Change living city of the 'Aroor' branch borrowers to Aroor.
19) Delete branches having deposit from Kollam.
20) Delete depositors of branches having number of customers between 1 and 3.
21) Delete depositors having deposit less than Rs500.

OUTPUT

1) Insert minimum 10 rows on each table and display that data

```
mysql> use vishnu
 Database changed
```

```
mysql> create table customer(Cust_id varchar(10) primary key,C_name
varchar(15) not null,City varchar(10));
Query OK, 0 rows affected (0.04 sec)
```

```
mysql>  create table branch(Branch_id varchar(5) primary key,B_name
varchar(15),City varchar(10));
Query OK, 0 rows affected (0.02 sec)
```

```
mysql>  create table deposit(Acc_no varchar(10) primary key,Cust_id
varchar(10) not null,Amount int,Branch_id varchar(5),oprn_date date);
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> create table borrow(loan_no varchar(5) primary key,cust_id
varchar(10),Branch_id varchar(5),Amount int);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> show tables;
+-----------------------+
| Tables_in_vishnu |
+-----------------------+
| borrow            |
| branch            |
| customer          |
| deposit           |
+-----------------------+
4 rows in set (0.00 sec)
```

```
mysql> insert into branch values
    -> (2001,'North','Cherthala'),
    -> (2002,'South','Cherthala'),
    -> (2004,'South','Aroor'),
    -> (2003,'North','Aroor');
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
mysql> select*from branch;
+--------------+-----------+--------------+
| Branch_id | B_name | city       |
+--------------+-----------+--------------+
| 2001       | North   | Cherthala |
| 2002       | South   | Cherthala |
| 2003       | North   | Aroor      |
| 2004       | South   | Aroor      |
+--------------+-----------+--------------+
4 rows in set (0.00 sec)
```

```
mysql> insert into deposit values
    -> (3001,1002,2002,20000,'2012-12-13'),
    -> (3002,1005,40000,2002,'2013-01-10'),
    -> (3003,1006,50000,2003,'2020-05-20'),
    -> (3004,1004,80000,2004,'2021-06-06'),
    -> (3005,1003,100000,2001,'2012-05-05'),
    -> (3006,1010,100,2003,'2017-08-11');
Query OK, 6 rows affected (0.01 sec)
Records: 6 Duplicates: 0  Warnings: 0
```

```
mysql> select*from deposit;
+-----------+------------+-----------+--------------+----------------+
| Acc_no | Cust_id | Amount | Branch_id | oprn_date  |
+-----------+------------+-----------+--------------+----------------+
| 3001    | 1002     |  20000 | 2002       | 2012-12-13 |
| 3002    | 1005     |  40000 | 2002       | 2013-01-10 |
| 3003    | 1006     |  50000 | 2003       | 2020-05-20 |
| 3004    | 1004     |  80000 | 2004       | 2021-06-06 |
| 3005    | 1003     | 100000 | 2001       | 2012-05-05 |
| 3006    | 1010     |    100 | 2003       | 2017-08-11 |
+-----------+------------+-----------+--------------+----------------+
6 rows in set (0.00 sec)
```

```
mysql> insert into borrow values
    -> (4001,1001,2004,10000),
    -> (4002,1007,2001,20000),
    -> (4003,1006,2003,30000),
    -> (4004,1009,2004,50000),
    -> (4005,1004,2001,15000);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

```
mysql> select*from borrow;
+------------+------------+--------------+-----------+
| loan_no | cust_id | Branch_id | Amount |
+------------+------------+--------------+-----------+
| 4001     | 1001     | 2004       |  10000 |
| 4002     | 1007     | 2001       |  20000 |
| 4003     | 1006     | 2003       |  30000 |
| 4004     | 1009     | 2004       |  50000 |
| 4005     | 1004     | 2001       |  15000 |
+------------+------------+--------------+-----------+
5 rows in set (0.00 sec)
```

```
mysql> insert into customer values
    -> (1001,'Sam','Kozhikode'),
    -> (1002,'Ashik','Kollam'),
    -> (1003,'Jithin','Malappuram'),
    -> (1004,'Vishnu','Thrissur'),
    -> (1005,'Prayag','Thrissur'),
    -> (1006,'Sooryan','Cherthala'),
    -> (1007,'Abhi','Cherthala'),
    -> (1008,'Ajay','Cherthala'),
    -> (1009,'Teena','Wayanad'),
    -> (1010,'Sulu','Aroor');
Query OK, 10 rows affected (0.01 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

```
mysql> select*from customer;
+------------+------------+----------------+
| Cust_id | C_name  | city        |
+------------+------------+----------------+
| 1001     | Sam      | Kozhikode  |
| 1002     | Ashik    | Kollam     |
| 1003     | Jithin   | Malappuram |
| 1004     | Vishnu   | Thrissur   |
| 1005     | Prayag   | Thrissur   |
| 1006     | Sooryan  | Cherthala  |
| 1007     | Abhi     | Cherthala  |
| 1008     | Ajay     | Cherthala  |
| 1009     | Teena    | Wayanad    |
| 1010     | Sulu     | Aroor      |
+------------+------------+----------------+
10 rows in set (0.00 sec)
```

2) List Cust_id along with customer name

```
mysql> select cust_id,c_name from customer;
+-----------+-----------+
| cust_id | c_name  |
+-----------+-----------+
| 1001    | Sam     |
| 1002    | Ashik   |
| 1003    | Jithin  |
| 1004    | Vishnu  |
| 1005    | Prayag  |
| 1006    | Sooryan |
| 1007    | Abhi    |
| 1008    | Ajay    |
| 1009    | Teena   |
| 1010    | Sulu    |
+-----------+-----------+
10 rows in set (0.00 sec)
```

3) List Cust_id of customers having amount greater than 10000

```
mysql> select cust_id,amount from deposit where amount>10000;
+-----------+-----------+
| cust_id | amount |
+-----------+-----------+
| 1002    |  20000 |
| 1005    |  40000 |
| 1006    |  50000 |
| 1004    |  80000 |
| 1003    | 100000 |
+-----------+-----------+
5 rows in set (0.00 sec)
```

4) List account date of customer 'sulu'

```
mysql> select open_date from deposit where cust_id=(select cust_id from
customer where c_name='sulu');
+----------------+
| open_date  |
+----------------+
| 2017-08-11 |
+----------------+
1 row in set (0.00 sec)
```

5) List Cust_id of customers who have opened account after 01/01/2016

- ```
  Mysql> select Cust_id from deposit where open_date>2016-01-01;
  +----------------+
  | Cust_id    |
  +----------------+
  | 1006       |
  | 1004       |
  | 1010       |
  +----------------+
  3 row in set (0.00 sec)
  ```

6) List account no. and Cust_id of customers having amount between 40,000 and 80,000

- ```
  Mysql> select Acc_no,Cust_id from deposit where amount between 40000
  and 80000;
  +-----------+------------+
  | acc_no | Cust_id |
  +-----------+------------+
  | 3002   |  1005   |
  | 3003   |  1006   |
  | 3004   |  1004   |
  +-----------+------------+
  3 row in set (0.00 sec)
  ```

7) List customer name starting with 'S'

- ```
  mysql> select c_name from customer where c_name like 's%';
  +------------+
  | c_name  |
  +------------+
  | sam     |
  | sooryan |
  | Sulu    |
  +------------+
  3 row in set (0.00 sec)
  ```

8) List customer from depositor starting with '_a%'

- ```
  mysql> select c_name from customer where Cust_id in(select cust_id from
  deposit where c_name like 's%');
  +------------+
  | c_name  |
  +------------+
  | jithin  |
  | Vishnu  |
  +------------+
  3 row in set (0.00 sec)
  ```

9) List customer name, account no and amount from deposit having exactly 5 characters in name

- ```
  mysql> select customer.c_name,deposit.acc_no,deposit.amount from customer,deposit where customer.cust_id=deposit.cust_id and length(customer.c_name)=5;
  +----------+----------+----------+
  | c_name | acc_no | amount |
  +----------+----------+----------+
  | Ashik  | 3001   |  20000 |
  +----------+----------+----------+
  1 row in set (0.00 sec)
  ```

10) List Cust_id, Loan no and Loan amount of borrowers

- ```
  mysql> select loan_no, cust_id, Branch_id from borrow;
  +------------+------------+--------------+
  | loan_no | cust_id | Branch_id |
  +------------+------------+--------------+
  | 4001    | 1001    | 2004      |
  | 4002    | 1007    | 2001      |
  | 4003    | 1006    | 2003      |
  | 4004    | 1009    | 2004      |
  | 4005    | 1004    | 2001      |
  +------------+------------+--------------+
  5 rows in set (0.00 sec)
  ```

11) List cust_id and C_name of depositors

- ```
  mysql> select c_name,Cust_id from customer where cust_id in (select Cust_id from deposit);
  +------------+------------+
  | Cust_id | C_name  |
  +------------+------------+
  | 1002    | Ashik   |
  | 1005    | Prayag  |
  | 1006    | Sooryan |
  | 1004    | Vishnu  |
  | 1003    | Jithin  |
  | 1010    | Sulu    |
  +------------+------------+
  □ rows in set (0.00 sec)
  ```
  ```

12) List all the customers who are depositors but not borrowers

```
mysql> select c_name from customer where cust_id in (select cust_id
from deposit) and cust_id not in (select cust_id from borrow);
+------------+
| c_name  |
+------------+
| Ashik   |
| Prayag  |
| Jithin  |
| Sulu    |
+------------+
```

13) List all the customers who are both depositors and borrowers

```
mysql> select c_name from customer where cust_id in (select cust_id
from deposit) and cust_id in (select cust_id from borrow);
+------------+
| c_name  |
+------------+
| Sooryan |
| Vishnu  |
+------------+
2 rows in set (0.01 sec)
```

14) List all the customers along with their amount who are either borrowers

```
mysql> select c_name from customer where cust_id in (select cust_id
from deposit) or cust_id in (select cust_id from borrow);
+------------+
| c_name  |
+------------+
| Sam     |
| Ashik   |
| Jithin  |
| Vishnu  |
| Prayag  |
| Sooryan |
| Abhi    |
| Teena   |
| Sulu    |
+------------+
9 rows in set (0.00 sec)
```

15) List the cites of depositor having branch 'Cherthala'

```
mysql> select city from customer where cust_id in (select cust_id
from deposit where branch_id in (select branch_id from branch where
b_name='cherthala'));
+----------------+
| city           |
+----------------+
| Malappuram |
+----------------+
1 row in set (0.00 sec)
```

16) Update 10% interest to all depositors

```
mysql> update deposit set amount=amount*1.1;
Query OK, 6 rows affected (0.01 sec)
Rows matched: 6  Changed: 6  Warnings: 0
```

```
mysql> select * from deposit;
+-----------+------------+-----------+---------------+-----------------+
| Acc_no | Cust_id | Amount | Branch_id | open_date  |
+-----------+------------+-----------+---------------+-----------------+
| 3001    | 1002     |  22000 | 2002       | 2012-12-13 |
| 3002    | 1005     |  44000 | 2002       | 2013-01-10 |
| 3003    | 1006     |  55000 | 2003       | 2020-05-20 |
| 3004    | 1004     |  88000 | 2004       | 2021-06-06 |
| 3005    | 1003     | 110000 | 2001       | 2012-05-05 |
| 3006    | 1010     |    110 | 2003       | 2017-08-11 |
+-----------+------------+-----------+---------------+-----------------+
6 rows in set (0.00 sec)
```

17) Update 10% to all depositors living in 'thrissur'

```
mysql> update deposit set amount=amount*1.1 where cust_id in(select
cust_id from customer where city='thrissur');
Query OK, 2 rows affected (0.01 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

```
mysql> select * from deposit;
+-----------+------------+-----------+---------------+-----------------+
| Acc_no | Cust_id | Amount | Branch_id | open_date  |
+-----------+------------+-----------+---------------+-----------------+
| 3001    | 1002     |  22000 | 2002       | 2012-12-13 |
| 3002    | 1005     |  48400 | 2002       | 2013-01-10 |
| 3003    | 1006     |  55000 | 2003       | 2020-05-20 |
| 3004    | 1004     |  96800 | 2004       | 2021-06-06 |
| 3005    | 1003     | 110000 | 2001       | 2012-05-05 |
| 3006    | 1010     |    110 | 2003       | 2017-08-11 |
+-----------+------------+-----------+---------------+-----------------+
6 rows in set (0.00 sec)
```

18) Change living city of the 'Aroor' branch borrowers to Aroor

```
mysql> update customer set city='Aroor' where cust_id in(select
cust_id from borrow where branch_id in (select branch_id from branch
where b_name='north'));
Query OK, 3 rows affected (0.01 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

```
mysql> select * from customer;
+------------+------------+----------------+
| Cust_id | C_name  | city        |
+------------+------------+----------------+
| 1001     | Sam      | Kozhikode  |
| 1002     | Ashik    | Kollam     |
| 1003     | Jithin   | Malappuram |
| 1004     | Vishnu   | Aroor      |
| 1005     | Prayag   | Thrissur   |
| 1006     | Sooryan  | Aroor      |
| 1007     | Abhi     | Aroor      |
| 1008     | Ajay     | Cherthala  |
| 1009     | Teena    | Wayanad    |
| 1010     | Sulu     | Aroor      |
+------------+------------+----------------+
10 rows in set (0.00 sec)
```

19) Delete branches having deposit from Kollam

```
mysql> delete from branch where branch_id in (select branch_id from
deposit where cust_id in (select cust_id from customer where
city='kollam'));
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from branch;
+--------------+-----------+--------------+
| Branch_id | B_name | city       |
+--------------+-----------+--------------+
| 2001       | North  | Cherthala |
| 2003       | North  | Aroor     |
| 2004       | South  | Aroor     |
+--------------+-----------+--------------+
3 rows in set (0.00 sec)
```

20) Delete depositors of branches having number of customers between 1 and 3

- ```
  Delete from deposit where Branch_id in(select Branch_id from branch
  where Branch_id in (select Branch_id from customer groupby Branch_id
  having count(Cust_id)Between 1 and 3));
  Query OK, 0 rows affected (0.92 sec)
  ```

- ```
  mysql> select * from deposit;
  +----------+-----------+----------+--------------+----------------+
  | Acc_no | Cust_id | Amount | Branch_id | open_date  |
  +----------+-----------+----------+--------------+----------------+
  | 3001   | 1002    |  22000 | 2002       | 2012-12-13 |
  | 3003   | 1006    |  55000 | 2003       | 2020-05-20 |
  | 3004   | 1004    |  96800 | 2004       | 2021-06-06 |
  | 3005   | 1003    | 110000 | 2001       | 2012-05-05 |
  +----------+-----------+----------+--------------+----------------+
  4 rows in set (0.00 sec)
  ```

21)Delete depositors having deposit less than Rs500

- ```
  mysql> delete from deposit where amount<500;
  Query OK, 1 row affected (0.01 sec)
  ```

- ```
  mysql> select * from deposit;
  +----------+-----------+----------+--------------+----------------+
  | Acc_no | Cust_id | Amount | Branch_id | open_date  |
  +----------+-----------+----------+--------------+----------------+
  | 3001   | 1002    |  22000 | 2002       | 2012-12-13 |
  | 3002   | 1005    |  48400 | 2002       | 2013-01-10 |
  | 3003   | 1006    |  55000 | 2003       | 2020-05-20 |
  | 3004   | 1004    |  96800 | 2004       | 2021-06-06 |
  | 3005   | 1003    | 110000 | 2001       | 2012-05-05 |
  +----------+-----------+----------+--------------+----------------+
  5 rows in set (0.00 sec)
  ```

Result: The program is executed successfully and the output is obtained.

Experiment no: 4

Aggregate and date function in SQL

Aim: Execute the aggregate functions like count, sum, avg etc. on the suitable database. Make use of built in functions according to the need of the database chosen. Retrieve the data from the database based on time and date functions like now (), date (), day (), time () etc. Use group by and having clauses.

Objective:

- To understand and implement various types of function in MYSQL.
- To learn the concept of group functions

NUMBER FUNCTION*:*

Abs(n): Select abs(-15) from dual;

Exp(n): Select exp(4) from dual;

Power(m,n): Select power(4,2) from dual;

Mod(m,n): Select mod(10,3) from dual;

Round(m,n): Select round(100.256,2) from dual;

Trunc(m,n): Select trunc(100.256,2) from dual;

Sqrt(m,n); Select sqrt(16) from dual;

Aggregate Functions:

1. **Count:** COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (*) indicates all the tuples of the column.

   Syntax: COUNT (Column name)
   Example: SELECT COUNT (Sal) FROM emp;

2. **SUM:** SUM followed by a column name returns the sum of all the values in that column. ***Syntax:*** SUM (Column name)

   Example: SELECT SUM (Sal) From emp;

3. **AVG:** AVG followed by a column name returns the average value of that column values. ***Syntax:*** AVG (n1, n2...)

   Example: Select AVG (10, 15, 30) FROM DUAL;

4. **MAX:** MAX followed by a column name returns the maximum value of that column. ***Syntax:*** MAX (Column name)

   Example: SELECT MAX (Sal) FROM emp;

mysql> select deptno, max(sal) from emp group by deptno;

DEPTNO	MAX (SAL)
10	5000
20	3000
30	2850

mysql> select deptno, max (sal) from emp group by deptno having max(sal)<3000;

DEPTNO	MAX(SAL)
30	2850

5. MIN: MIN followed by column name returns the minimum value of that column.

Syntax: MIN (Column name)

Example: SELECT MIN (Sal) FROM emp; mysql> select deptno,min(sal) from emp group by deptno having min(sal)>1000;

DEPTNO	MIN (SAL)
10	1300

DATE FUNCTIONS:

CURDATE()

Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

mysql> SELECT CURDATE();

CURDATE()
2025-05-25

1 row in set (0.00 sec)

CURTIME()

Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT CURTIME();

+---------------+
| CURTIME()  |
+---------------+
| 14:45:23       |
+---------------+

1 row in set (0.00 sec)
```

DATE(expr)

Extracts the date part of the date or datetime expression expr.

```
mysql> SELECT DATE('2025-05-26 14:30:00');

+-----------------------------------+
| DATE('2025-05-26 14:30:00') |
+-----------------------------------+
| 2025-05-26                        |
+-----------------------------------+

1 row in set (0.00 sec)
```

DAYNAME(date)

Returns the name of the weekday for date.

```
msql> SELECT DAYNAME('2025-05-26');

+-------------------------------+
| DAYNAME('2025-05-26') |
+-------------------------------+
| Monday                        |
+-------------------------------+

1 row in set (0.00 sec)
```

HOUR(time)

Returns the hour for the time. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');

+---------------------+
| HOUR('10:05:03') |
+---------------------+
|                   10 |
+---------------------+

1 row in set (0.00 sec)
```

LAST_DAY(date)

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

```
mysql> SELECT LAST_DAY('2025-05-12');

+--------------------------------+
| LAST_DAY('2025-05-12') |
+--------------------------------+
| 2025-05-31                     |
+--------------------------------+

1 row in set (0.00 sec)
```

MINUTE(time)

Returns the minute for time, in the range 0 to 59.

```
mysql> SELECT MINUTE('10:25:45');

+------------------------+
| MINUTE('10:25:45') |
+------------------------+
|                     25 |
+------------------------+

1 row in set (0.00 sec)
```

MONTH(date)

Returns the month for date, in the range 0 to 12.

```
mysql> SELECT MONTH('1998-02-03')

+----------------------------+
| MONTH('2025-05-26')  |
+----------------------------+
|                          5 |
+----------------------------+

1 row in set (0.00 sec)
```

MONTHNAME(date)

Returns the full name of the month from a given date value as a string.

```
mysql> SELECT MONTHNAME('2025-05-26');

+-----------------------------------+
| MONTHNAME('2025-05-26') |
+-----------------------------------+
| May                               |
+-----------------------------------+

1 row in set (0.00 sec
```

NOW()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT NOW();

+-------------------------+
| NOW()                   |
+-------------------------+
| 2025-05-26 14:45:12 |
+-------------------------+

1 row in set (0.00 sec)
```

GROUP BY: This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation.

Syntax: SELECT <set of fields> FROM <relation_name> GROUP BY <field_name>;

Example: SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY EMPNO;

GROUP BY-HAVING : The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Syntax: SELECT column_name, aggregate_function(column_name) FROM table_name WHERE column_name operator value GROUP BY column_name HAVING aggregate_function(column_name) operator value;

Example: SELECT empno,SUM(SALARY) FROM emp,dept WHERE emp.deptno =20 GROUP BY empno;

ORDER BY: This query is used to display a selected set of fields from a relation in an ordered manner base on some field.

Syntax: SELECT <set of fields> FROM <relation_name> ORDER BY <field_name>;

Example: SQL> SELECT empno, ename, job FROM emp ORDER BY job;

LAB PRACTICE ASSIGNMENT:

Consider the following table structure for this assignment:

CUSTOMER(Cust_id, C_name, City)

BRANCH(Branch_id, bname, City)

DEPOSIT(Acc_no , Cust_id, Amount, Branch_id, Open_date)

BORROW(Loan_no, Cust_id, Branch_id, Amount)

Perform the following queries on the above table:

1) List total loan.
2) List total deposit.
3) List maximum deposit of customers living in Ernakulam.
4) Count total number of branch cities.
5) List branch_id and branch wise deposit.
6) List the branches having sum of deposit more than 4000.
7) List the names of customers having minimum deposit.
8) Count the number of depositors living in 'Cherthala'.
9) Find out number of customers living in Kozhikode.
10) Find the maximum deposit of the Kozhikode branch.
11) Find out the customers who are not living in Ernakulam or Alappuzha.
12) List out Cust_id and C_name in descending order of their C_name.
13) Display the number of depositors in branch wise.
14) Find out the branch which has not borrowers.
15) How many customers have opened deposit after '01-01-2016'

OUTPUT

Consider the following table structure for this assignment:

```
mysql> use bank;
Database changed

mysql> create table customerr(cust_id varchar(5),c_name
varchar(10),city varchar(10));
Query OK, 0 rows affected (0.48 sec)

mysql> insert into customerr values('c02','manu','aroor');
Query OK, 1 row affected (0.06 sec)

mysql> insert into customerr values('c04','varun','cherthala');
Query OK, 1 row affected (0.08 sec)

mysql> insert into customerr values('c05','arun','thrissur');
Query OK, 1 row affected (0.07 sec)

mysql> insert into customerr values('c06','anu','ktm');
Query OK, 1 row affected (0.06 sec)

mysql> insert into customerr values('c09','ajay','delhi');
Query OK, 1 row affected (0.06 sec)

mysql> insert into customerr values('c10','diya','goa');
Query OK, 1 row affected (0.09 sec)

mysql> create table branch(b_id varchar(5),bname varchar(10),city
varchar(10));
Query OK, 0 rows affected (0.80 sec)

mysql> insert into branch values('b01','alpy','alappuzha');
Query OK, 1 row affected (0.06 sec)

mysql> insert into branch values('b02','aroor','pallipuram');
Query OK, 1 row affected (0.07 sec)

mysql> insert into branch values('b03','aluva','ekm');
Query OK, 1 row affected (0.08 sec)

mysql> insert into branch values('b04','palayam','tvm');
Query OK, 1 row affected (0.09 sec)

mysql> insert into branch values('b07','mysoor','karnataka');
Query OK, 1 row affected (0.06 sec)
```

```
mysql> insert into branch values('b06','delhi','delhi');
Query OK, 1 row affected (0.11 sec)

mysql> create table deposit(acc_no varchar(10),cust_id
varchar(5),amt int,branch_id varchar(5),opendate date);
Query OK, 0 rows affected (0.44 sec)

mysql> insert into deposit values('a21','c03',45000,'b33','2019-10-
01');
Query OK, 1 row affected (0.08 sec)

mysql> insert into deposit values('a34','c02',67000,'b09','2016-01-
01');
Query OK, 1 row affected (0.06 sec)

mysql> insert into deposit values('a03','c06',7000,'b10','2026-12-
09');
Query OK, 1 row affected (0.06 sec)

mysql> insert into deposit values('a06','c45',11000,'b87','2002-04-
21');
Query OK, 1 row affected (0.08 sec)

mysql> create table borrow(loan_no varchar(5),cust_id
varchar(5),branch_id varchar(5),amt int);
Query OK, 0 rows affected (0.43 sec)

mysql> insert into borrow values('002','c02','b23',2300);
Query OK, 1 row affected (0.09 sec)

mysql> insert into borrow values('010','c10','b33',9300);
Query OK, 1 row affected (0.08 sec)

mysql> insert into borrow values('023','c09','b04',6700);
Query OK, 1 row affected (0.06 sec)

mysql> insert into borrow values('078','c07','b05',34500);
Query OK, 1 row affected (0.07 sec)

mysql> insert into borrow values('034','c25','b01',4567);
Query OK, 1 row affected (0.08 sec)
```

- mysql> select * from customerr;

```
+------------+-----------+--------------+
| cust_id | c_name | city       |
+------------+-----------+--------------+
| c02      | manu    | aroor      |
| c04      | varun   | cherthala |
| c05      | arun    | thrissur   |
| c06      | anu     | ktm        |
| c09      | ajay    | delhi      |
| c10      | diya    | goa        |
+------------+-----------+--------------+
```

- rows in set (0.00 sec)

- mysql> select * from branch;

```
+--------+------------+----------------+
| b_id | bname    | city        |
+--------+------------+----------------+
| b01   | alpy     | alappuzha   |
| b02   | aroor    | pallipuram |
| b03   | aluva    | ekm         |
| b04   | palayam | tvm         |
| b07   | mysoor   | karnataka   |
| b06   | delhi    | delhi       |
+--------+------------+----------------+
6 rows in set (0.00 sec)
```

- mysql> select * from deposit;

```
+-----------+------------+-----------+---------------+-----------------+
| acc_no | cust_id | amt     | branch_id | opendate    |
+-----------+------------+-----------+---------------+-----------------+
| a01     | c01      | 100000 | b03        | 2023-10-10 |
| a21     | c03      |  45000 | b33        | 2019-10-01 |
| a34     | c02      |  67000 | b09        | 2016-01-01 |
| a03     | c06      |   7000 | b10        | 2026-12-09 |
| a06     | c45      |  11000 | b87        | 2002-04-21 |
+-----------+------------+-----------+---------------+-----------------+
5 rows in set (0.00 sec)
```

- mysql> select * from borrow;

```
+------------+------------+---------------+----------+
| loan_no | cust_id | branch_id | amt    |
+------------+------------+---------------+----------+
| 002      | c02      | b23        |  2300 |
| 010      | c10      | b33        |  9300 |
| 023      | c09      | b04        |  6700 |
| 078      | c07      | b05        | 34500 |
| 034      | c25      | b01        |  4567 |
+------------+------------+---------------+----------+
rows in set (0.00 sec)
```

1) List total loan

```
mysql> select sum(amt) as totalloan from borrow;
+--------------+
| totalloan |
+--------------+
|     57367 |
+--------------+
1 row in set (0.01 sec)
```

2) List total deposit

```
mysql> select sum(amt) as totaldeposit from deposit;
+-------------------+
| totaldeposit |
+-------------------+
|       230000 |
+-------------------+
1 row in set (0.00 sec)
```

3) List maximum deposit of customers living in Ernakulam

```
mysql> select max(amt) as maxdeposit from deposit where cust_id
in(select cust_id from customer where city='ekm');
+----------------+
| maxdeposit |
+----------------+
|       NULL |
+----------------+
1 row in set (0.00 sec)
```

4) Count total number of branch cities

```
mysql> select count(distinct city)as totalbranchcities from branch;
+-------------------------+
| totalbranchcities |
+-------------------------+
|                   6 |
+-------------------------+
1 row in set (0.00 sec)
```

5) List branch_id and branch wise deposit

```
mysql> select branch_id,sum(amt)as totaldeposit from deposit group
by branch_id;
+--------------+------------------+
| branch_id | totaldeposit |
+--------------+------------------+
| b03       |       100000 |
| b33       |        45000 |
| b09       |        67000 |
| b10       |         7000 |
| b87       |        11000 |
+--------------+------------------+
5 rows in set (0.00 sec)
```

6) How many customers have opened deposit after '01-01-2016'

```
mysql> select count(distinct cust_id)as customercount from deposit
where opendate >'2016-01-01';
+--------------------+
| customercount |
+--------------------+
|             3 |
+--------------------+
1 row in set (0.00 sec)
```

7) List the branches having sum of deposit more than 4000

```
mysql> select branch_id,sum(amt) as total_deposit from deposit group
by branch_id having sum(amt)>4000;
+--------------+--------------------+
| branch_id | total_deposit |
+--------------+--------------------+
| b03       |        100000 |
| b33       |         45000 |
| b09       |         67000 |
| b10       |          7000 |
| b87       |         11000 |
+--------------+--------------------+
5 rows in set (0.00 sec)
```

8) List the names of customers having minimum deposit

```
mysql> select c_name from customerr where cust_id in(select cust_id
from deposit group by cust_id having min(amt)=(select min(amt) from
deposit));
+-----------+
| c_name |
+-----------+
| anu    |
+-----------+
1 row in set (0.00 sec)
```

9) Count the number of depositors living in 'Kozhikode'

```
mysql> select count(distinct cust_id)as countdepositors from deposit
where cust_id in (select cust_id from customerr where city='ktm');
+----------------------+
| countdepositors |
+----------------------+
|               1 |
+----------------------+
1 row in set (0.00 sec)
```

10) Find the maximum deposit of the kozhikode branch

```
mysql> select max(amount) as maxdeposit from deposite where cust_id
in (select cust_id from customer where city = 'kozhikode');
+----------------+
| maxdeposit |
+----------------+
|      100000|
+----------------+
1 row in set (0.03 sec)
```

11) Find out number of customers living in Ernakulam

```
mysql> select count(*) as count_customer from customer where
city='ekm';
+---------------------+
| count_customer |
+---------------------+
|              0 |
+---------------------+
1 row in set (0.00 sec)
```

12) Find out the customers who are not living in Ernakulam or Alappuzha

```
mysql> select * from customerr where city not in ('ekm','ktm');
+------------+-----------+--------------+
| cust_id | c_name | city       |
+------------+-----------+--------------+
| c02      | manu    | aroor      |
| c04      | varun   | cherthala |
| c05      | arun    | thrissur   |
| c09      | ajay    | delhi      |
| c10      | diya    | goa        |
+------------+-----------+--------------+
5 rows in set (0.00 sec)
```

13) List out Cust_id and C_name in descending order of their C_name

```
mysql> select cust_id,c_name from customerr order by c_name desc;
```

```
+------------+-----------+
| cust_id | c_name |
+------------+-----------+
| c04      | varun  |
| c02      | manu   |
| c10      | diya   |
| c05      | arun   |
| c06      | anu    |
| c09      | ajay   |
+------------+-----------+
6  rows in set (0.00 sec)
```

14) Display the number of depositors in branch wise

- mysql> select branch_id,count(distinct cust_id)as countdepositor from deposit group by branch_id;

```
+---------------+---------------------+
| branch_id | countdepositor |
+---------------+---------------------+
| b03       |               1 |
| b09       |              1 |
| b10       |               1 |
| b33       |               1 |
| b87       |                1 |
+  ---------------+---------------------+
  5 rows in set (0.00 sec)
```

15) Find out the branch which has not borrowers

- mysql> select * from branch where b_id not in(select distinct branch_id from borrow);

```
+--------+-----------+----------------+
| b_id | bname  | city        |
+--------+-----------+----------------+
| b02  | aroor  | pallipuram |
| b03  | aluva  | ekm         |
| b07  | mysoor | karnataka  |
| b06  | delhi  | delhi       |
+--------+-----------+----------------+
4 rows in set (0.00 sec)
```

Result: The program is executed successfully and the output is obtained.

Experiment no: 5

Types of join in SQL

Aim: Implement nested sub queries. Perform a test for set membership (in, not in), set comparison (<some, >=some, <all etc.) and set cardinality (unique, not unique).

Objective:

- To learn different types of Joins.
- To implement different sub queries.

Theory :

MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.

There are three types of MySQL joins:

- MySQL INNER JOIN (or sometimes called simple join)
- MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

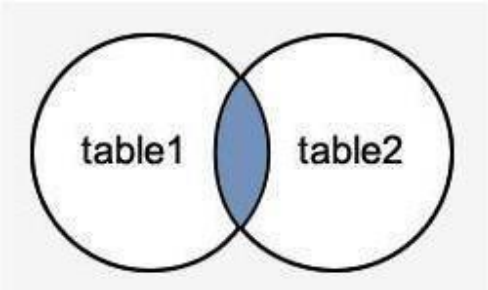
MySQL Inner JOIN (Simple Join)

The MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Image representation:



Let's take an example:

Consider two tables "officers" and "students", having the following data. **Execute the following query:**

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
INNER JOIN students
ON officers.officer_id = students.student_id;
```

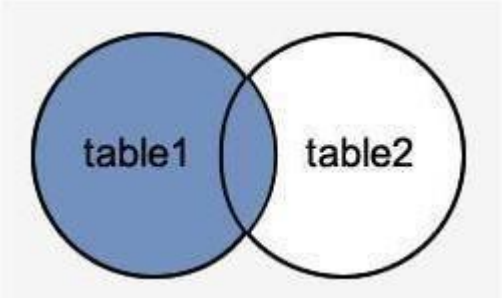
MySQL Left Outer Join

The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

Syntax:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Image representation:



Let's take an example:

Consider two tables "officers" and "students", having the following data. **Execute the following query:**

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
LEFT JOIN students
ON officers.officer_id = students.student_id;
```

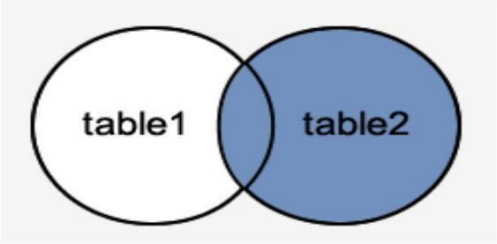
MySQL Right Outer Join

The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where he join condition is fulfilled.

Syntax:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Image representation:



Let's take an example:

Consider two tables "officers" and "students", having the following data. **Execute the following query:**

```
SELECT  officers.officer_name, officers.address, students.course_name, students.student_name
FROM officers
RIGHT JOIN students
ON officers.officer_id = students.student_id;
```

SPECIAL OPERATOR:

MySQL IN Condition

The MySQL IN condition is used to reduce the use of multiple OR conditions in a SELECT, INSERT, UPDATE and DELETE statement.

Syntax:

expression IN (value1, value2,.....value_n);

Parameters:

expression: It specifies a value to test.

value1, value2, ...or value_n: These are the values to test against expression. If any of these values matches expression, then the IN condition will evaluate to true. This is a quick method to test if any one of the values matches expression.

Execute the following query:

```
SELECT *
FROM officers
WHERE officer_name IN ('Ajeet', 'Vimal', 'Deepika');
```

MySQL NOT Condition

The MySQL NOT condition is opposite of MySQL IN condition. It is used to negate a condition in a SELECT, INSERT, UPDATE or DELETE statement.

Syntax:

NOT condition

Parameter:

condition: It specifies the conditions that you want to negate.

MySQL NOT Operator with IN condition

Consider a table "officers", having the following data.

Execute the following query:

```
SELECT *
FROM officers
WHERE officer_name NOT IN ('Ajeet','Vimal','Deepika');
```

MySQL IS NULL Condition

MySQL IS NULL condition is used to check if there is a NULL value in the expression. It is used with SELECT, INSERT, UPDATE and DELETE statement.

Syntax:

expression IS NULL

Parameter:
expression: It specifies a value to test if it is NULL
Execute the following query:

```
SELECT *
FROM officers
WHERE officer_name IS NULL;
```

MySQL IS NOT NULL Condition

MySQL IS NOT NULL condition is used to check the NOT NULL value in the expression. It is used with SELECT, INSERT, UPDATE and DELETE statements.

Syntax:
expression IS NOT NULL

Parameter:

expression: It specifies a value to test if it is not NULL value.
Execute the following query:

```
SELECT *
FROM officers
WHERE officer_name IS NOT NULL;
```

SET OPERATORS:

The Set operator combines the result of 2 queries into a single result. The following are the operators:

- Union
- Union all
- Intersect
- Minus

LAB PRACTICE ASSIGNMENT:

Consider the following table structure for this assignment:

- Location(Location_Id integer, Reginal_Group varchar(20))
- Department (Department_Id, Name, Location_Id)
- Job(Job_Id Integer,Function Varchar(30))
- Employee(Employee_Id, Lastname ,Firstname, Middlename, Job_Id, Manager_Id, Hiredate, Salary, Department_Id)
- Loan(Employee_Id, Firstname , Loan_Amount)

Perform the following queries on the above table:

1) Perform all types of JOIN operations on Employee and Loan tables.
2) Perform all types of set operations on Employee and Loan tables.
3) Find out no.of employees working in "Sales" department
4) Find out the employees who are not working in department 202 or 205.
5) List out employee id, last name in descending order based on the salary column.
6) How many employees who are working in different departments wise in the organization
7) List out the department id having at least four employees **8)** Display the employee who got the maximum salary.
9) Update the employees' salaries, who are working as Clerk on the basis of 10%.
10) Delete the employees who are working in accounting department.
11) Find out whose department has not employees.
12) List out the department wise maximum salary, minimum salary, average salary of the employees
13) How many employees who are joined in 2022.
14) Display the employees who are working in "south"
15) List our employees with their department names

OUTPUT

```
mysql> CREATE TABLE Location (Location_Id INTEGER PRIMARY
KEY,Reginal_Group VARCHAR(20));
```

```
mysql> CREATE TABLE Department (Department_Id INTEGER PRIMARY KEY,Name
VARCHAR(255), Location_Id INTEGER,FOREIGN KEY (Location_Id) REFERENCES
Location(Location_Id));
```

```
mysql> CREATE TABLE Job (Job_Id INTEGER PRIMARY KEY, Funct
VARCHAR(30));
```

```
mysql>CREATE TABLE Employee (Employee_Id INTEGER PRIMARY KEY, Lastname
VARCHAR(255), Firstname VARCHAR(255), Middlename VARCHAR(255), Job_Id
INTEGER, Manager_Id INTEGER, Hiredate DATE, Salary DECIMAL(10, 2),
Department_Id INTEGER, FOREIGN KEY (Job_Id) REFERENCES Job(Job_Id),
FOREIGN KEY (Manager_Id) REFERENCES Employee(Employee_Id), FOREIGN KEY
(Department_Id) REFERENCES Department(Department_Id));
```

```
mysql> CREATE TABLE Loan (Employee_Id INTEGER, Firstname VARCHAR(255),
Loan_Amount DECIMAL(10, 2), FOREIGN KEY (Employee_Id) REFERENCES
Employee(Employee_Id));
```

- **Insert sample data into Location table**

```
mysql> INSERT INTO Location (Location_Id, Reginal_Group) VALUES (101,
'North'), (102, 'South'), (103, 'East'), (104, 'West');
```

```
mysql> SELECT * FROM Location;
+-----------------+--------------------+
| Location_Id | Reginal_Group |
+-----------------+--------------------+
|         101 | North          |
|         102 | South          |
|         103 | East           |
|         104 | West           |
+-----------------+--------------------+
4 rows in set (0.00 sec)
```

- **Insert sample data into Department table**

```
mysql> INSERT INTO Department (Department_Id, Name, Location_Id) VALUES
(201, 'IT', 101), (202, 'HR', 102), (203, 'Finance', 103), (204,
'Marketing', 104), (205, 'Operations', 101), (206, 'Sales', 102), (207,
'Research', 103), (208, 'Production', 104);
```

```
mysql> SELECT * FROM Department;
+---------------+----------+-------------+
| Department_Id | Name     | Location_Id |
+---------------+----------+-------------+
|           201 | IT       |         101 |
|           202 | HR       |         102 |
|           203 | Finance  |         103 |
|           204 | Marketing|         104 |
|           205 | Operations|        101 |
|           206 | Sales    |         102 |
|           207 | Research |         103 |
|           208 | Production|        104 |
+---------------+----------+-------------+
8 rows in set (0.00 sec)
```

- **Insert sample data into Job table**

```
mysql> INSERT INTO Job (Job_Id, Funct) VALUES (301, 'Manager'), (302,
'Developer'), (303, 'Analyst'), (304, 'HR Manager'), (305, 'Research
Analyst');

mysql> SELECT * FROM Job;
+--------+-------------------+
| Job_Id | Funct             |
+--------+-------------------+
|    301 | Manager           |
|    302 | Developer         |
|    303 | Analyst           |
|    304 | HR Manager        |
|    305 | Research Analyst  |
+--------+-------------------+
5 rows in set (0.00 sec)
```

- **Insert sample data into Employee table**

```
mysql> INSERT INTO Employee (Employee_Id, Lastname, Firstname,
Middlename, Job_Id, Manager_Id, Hiredate, Salary, Department_Id) VALUES
 (401, 'M S', 'Vishnu',NULL , 301, NULL, '2022-01-01', 50000.00, 202),
 (402, 'R S', 'Lijith', NULL, 301, NULL, '2022-02-01', 40000.00, 202),
 (403, 'Simon', 'Sam', NULL, 302, 401, '2022-03-01', 45000.00, 205),
 (404, 'Joby', 'Aljo', 'Emmanual', 302, 402, '2022-04-01', 55000.00,
207), (405, 'T', 'Jithin', 'Kuruvila', 303, 401, '2022-05-01'60000.00,
206), (406, 'Davis', 'Tessa', 'Poonjar', 303, 402, '2022-06-
01'65000.00, 207), (407, 'Miller', 'David', 'Jhon', 304, 401, '2022-07-
01', 70000.00, 208), (408, 'Kombanayil', 'Mohanlal', 'John', 304, 402,
'2022-08-01', 75000.00, 201),(409, 'Manadhavady', 'Teena', 'Mathew',
305, 401, '2022-09-01', 80000.00, 203), (410, 'Kharim', 'Ashik', 'K',
305, 402, '2022-10-01', 85000.00, 204);

mysql> SELECT * FROM Employee;
```

Employee_Id	Lastname	Firstname	Middlename	Job_Id	Manager_Id	Hiredate	Salary	Department_Id
401	M S	Vishnu	NULL	301	NULL	2022-01-01	50000.00	202
402	R S	Lijith	NULL	301	NULL	2022-02-01	40000.00	202
403	Simon	Sam	NULL	302	401	2022-03-01	45000.00	205
404	Joby	Aljo	Emmanual	302	402	2022-04-01	55000.00	207
405	T	Jithin	Kuruvila	303	401	2022-05-01	60000.00	206
406	Davis	Tessa	Poonjar	303	402	2022-06-01	65000.00	207
407	Miller	David	Jhon	304	401	2022-07-01	70000.00	208
408	Kombanayil	Mohanlal	John	304	402	2022-08-01	75000.00	201
409	Manadhavady	Teena	Mathew	305	401	2022-09-01	80000.00	203
410	Kharim	Ashik	K	305	402	2022-10-01	85000.00	204

- **Insert sample data into Loan table**

```
mysql> INSERT INTO Loan (Employee_Id, Firstname, Loan_Amount) VALUES
(401, 'Vishnu', 100000.00),(402, 'Lijith', 2000000.00),(403, 'Sam',
15000.00),(404, 'Aljo', 250000.00),(405, 'Jithin', 33000.00);
```

```
mysql> SELECT * FROM Loan;
+-----------------+--------------+-----------------+
| Employee_Id | Firstname | Loan_Amount |
+-----------------+--------------+-----------------+
|         401 | Vishnu    |   100000.00 |
|         402 | Lijith    |  2000000.00 |
|         403 | Sam       |    15000.00 |
|         404 | Aljo      |   250000.00 |
|         405 | Jithin    |    33000.00 |
+-----------------+--------------+-----------------+
5 rows in set (0.00 sec)
```

1) Perform all types of JOIN operations on Employee and Loan tables

- **a) Inner Join**

```
mysql> SELECT Employee.Employee_Id, Employee.Firstname,
Loan.Loan_Amount FROM Employee INNER JOIN Loan ON Employee.Employee_Id
= Loan.Employee_Id;
+-----------------+--------------+-----------------+
| Employee_Id | Firstname | Loan_Amount |
+-----------------+--------------+-----------------+
|         401 | Vishnu    |   100000.00 |
|         402 | Lijith    |  2000000.00 |
|         403 | Sam       |    15000.00 |
|         404 | Aljo      |   250000.00 |
|         405 | Jithin    |    33000.00 |
+-----------------+--------------+-----------------+
5 rows in set (0.00 sec)
```

- **b)Left Join**

```
mysql> SELECT Employee.Employee_Id, Employee.Firstname,
Loan.Loan_Amount FROM Employee LEFT JOIN Loan ON Employee.Employee_Id =
Loan.Employee_Id;

+-------------+-----------+-------------+
| Employee_Id | Firstname | Loan_Amount |
+-------------+-----------+-------------+
|         401 | Vishnu    |   100000.00 |
|         402 | Lijith    |  2000000.00 |
|         403 | Sam       |    15000.00 |
|         404 | Aljo      |   250000.00 |
|         405 | Jithin    |    33000.00 |
|         406 | Tessa     |        NULL |
|         407 | David     |        NULL |
|         408 | Mohanlal  |        NULL |
|         409 | Teena     |        NULL |
|         410 | Ashik     |        NULL |
+-------------+-----------+-------------+
10 rows in set (0.00 sec);
```

- **c)Right Join**

```
mysql> SELECT Employee.Employee_Id, Employee.Firstname,
Loan.Loan_Amount FROM Employee RIGHT JOIN Loan ON Employee.Employee_Id
= Loan.Employee_Id;

+-------------+-----------+-------------+
| Employee_Id | Firstname | Loan_Amount |
+-------------+-----------+-------------+
|         401 | Vishnu    |   100000.00 |
|         402 | Lijith    |  2000000.00 |
|         403 | Sam       |    15000.00 |
|         404 | Aljo      |   250000.00 |
|         405 | Jithin    |    33000.00 |
+-------------+-----------+-------------+
5 rows in set (0.00 sec);
```

2) Perform all types of set operations on Employee and Loan tables

- **a)Union**

```
mysql> SELECT Employee_Id, Lastname, Firstname FROM Employee UNION
SELECT Employee_Id, Firstname, NULL FROM Loan;
```

```
+-----------------+-----------------+--------------+
| Employee_Id | Lastname     | Firstname |
+-----------------+-----------------+--------------+
|         401 | M S          | Vishnu    |
|         402 | R S          | Lijith    |
|         403 | Simon        | Sam       |
|         404 | Joby         | Aljo      |
|         405 | T            | Jithin    |
|         406 | Davis        | Tessa     |
|         407 | Miller       | David     |
|         408 | Kombanayil   | Mohanlal  |
|         409 | Manadhavady  | Teena     |
|         410 | Kharim       | Ashik     |
|         401 | Vishnu       | NULL      |
|         402 | Lijith       | NULL      |
|         403 | Sam          | NULL      |
|         404 | Aljo         | NULL      |
|         405 | Jithin       | NULL      |
+-----------------+-----------------+--------------+
15 rows in set (0.00 sec);
```

- **b)Intersection**

```
mysql> SELECT Employee_Id, Firstname FROM Employee INTERSECT SELECT
Employee_Id, Firstname FROM Loan;
+-----------------+--------------+
| Employee_Id | Firstname |
+-----------------+--------------+
|         401 | Vishnu    |
|         402 | Lijith    |
|         403 | Sam       |
|         404 | Aljo      |
|         405 | Jithin    |
+-----------------+--------------+
5 rows in set (0.00 sec);
```

- **c)Difference**

```
mysql> SELECT Employee_Id, Firstname FROM Employee EXCEPT SELECT
Employee_Id, Firstname FROM Loan;
+-----------------+--------------+
| Employee_Id | Firstname |
+-----------------+--------------+
|         406 | Tessa     |
|         407 | David     |
|         408 | Mohanlal  |
|         409 | Teena     |
|         410 | Ashik     |
+-----------------+--------------+
5 rows in set (0.00 sec);
```

3) Find out no.of employees working in "Sales" department

```
mysql> SELECT COUNT(*) AS EmployeeCount FROM Employee JOIN Department
ON Employee.Department_Id = Department.Department_Id WHERE
Department.Name = 'Sales';
+--------------------+
| EmployeeCount |
+--------------------+
|              1 |
+--------------------+
1 rows in set (0.00 sec);
```

4) Find out the employees who are not working in department 202 or 205

```
mysql> SELECT Employee_Id,Lastname, Firstname FROM Employee WHERE
Department_Id NOT IN (202,205);
+-----------------+-----------------+---------------+
| Employee_Id | Lastname     | Firstname |
+-----------------+-----------------+---------------+
|          408 | Kombanayil   | Mohanlal  |
|          409 | Manadhavady | Teena      |
|          410 | Kharim       | Ashik      |
|          405 | T            | Jithin     |
|          404 | Joby         | Aljo       |
|          406 | Davis        | Tessa      |
|          407 | Miller       | David      |
+-----------------+-----------------+---------------+
7 rows in set (0.00 sec);
```

5) List out employee id, last name in descending order based on the salary column

```
mysql> SELECT Employee_Id, Firstname FROM Employee ORDER BY Salary
DESC;
+-----------------+---------------+
| Employee_Id | Firstname |
+-----------------+---------------+
|          410 | Ashik      |
|          409 | Teena      |
|          408 | Mohanlal   |
|          407 | David      |
|          406 | Tessa      |
|          405 | Jithin     |
|          404 | Aljo       |
|          401 | Vishnu     |
|          403 | Sam        |
|          402 | Lijith     |
+-----------------+---------------+
10 rows in set (0.00 sec);
```

6) How many employees who are working in different departments wise in the organization

```
mysql> SELECT Department.Name, COUNT(*) AS EmployeeCount FROM Employee
JOIN Department ON Employee.Department_Id = Department.Department_Id
GROUP BY Department.Name;
+----------------+--------------------+
| Name           | EmployeeCount |
+----------------+--------------------+
| IT             |                  1 |
| HR             |                  2 |
| Finance        |                  1 |
| Marketing      |                  1 |
| Operations |                  1 |
| Sales          |                  1 |
| Research       |                  2 |
| Production |                  1 |
+----------------+--------------------+
8 rows in set (0.00 sec);
```

7) List out the department id having at least Two employees

```
mysql> SELECT Department_Id FROM Employee GROUP BY Department_Id HAVING
COUNT(*) >= 2;
+--------------------+
| Department_Id |
+--------------------+
|            202 |
|            207 |
+--------------------+
2 rows in set (0.00 sec);
```

8) Display the employee who got the maximum salary

```
mysql> SELECT COUNT(*) FROM Employee WHERE YEAR(Hiredate) = 1985;
mysql> SELECT Employee_Id,Firstname FROM Employee WHERE Salary =
(SELECT MAX(Salary) FROM Employee);
+-----------------+--------------+
| Employee_Id | Firstname |
+-----------------+--------------+
|          410 | Ashik      |
+-----------------+--------------+
1 rows in set (0.00 sec);
```

9) Update the employees' salaries, who are working as Clerk on the basis of 10%

```
mysql> UPDATE Employee SET Salary = Salary * 1.1 WHERE Job_Id = (SELECT
Job_Id FROM Job WHERE `Funct` = 'Manager');
Query OK, 2 rows affected (0.40 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

- mysql> SELECT Employee.Employee_Id, Employee.Firstname,Job.Funct,Employee.Salary FROM Employee INNER JOIN Job ON Employee.Job_Id = Job.Job_Id;

```
+-------------+-----------+----------------------+-----------+
| Employee_Id | Firstname | Funct                | Salary    |
+-------------+-----------+----------------------+-----------+
|         401 | Vishnu    | Manager              | 55000.00 |
|         402 | Lijith    | Manager              | 44000.00 |
|         403 | Sam       | Developer            | 45000.00 |
|         404 | Aljo      | Developer            | 55000.00 |
|         405 | Jithin    | Analyst              | 60000.00 |
|         406 | Tessa     | Analyst              | 65000.00 |
|         407 | David     | HR Manager           | 70000.00 |
|         408 | Mohanlal  | HR Manager           | 75000.00 |
|         409 | Teena     | Research Analyst     | 80000.00 |
|         410 | Ashik     | Research Analyst     | 85000.00 |
+-------------+-----------+----------------------+-----------+
10 rows in set (0.00 sec);
```

10) Delete the employees who are working in accounting department

- mysql> DELETE FROM Employee WHERE Department_Id = (SELECT Department_Id FROM Department WHERE Name ='IT');
  Query OK, 1 row affected (0.08 sec)

- mysql> SELECT Employee.Employee_Id, Employee.Firstname,Department.Name FROM Employee INNER JOIN Department ON Employee.Department_Id = Department.Department_Id;

```
+-------------+-----------+------------+
| Employee_Id | Firstname | Name       |
+-------------+-----------+------------+
|         401 | Vishnu    | HR         |
|         402 | Lijith    | HR         |
|         403 | Sam       | Operations |
|         404 | Aljo      | Research   |
|         405 | Jithin    | Sales      |
|         406 | Tessa     | Research   |
|         407 | David     | Production |
|         409 | Teena     | Finance    |
|         410 | Ashik     | Marketing  |
+-------------+-----------+------------+
10 rows in set (0.00 sec);
```

11) Find out whose department has not employees

```
mysql> SELECT Department.Department_Id, Department.Name FROM Department
LEFT JOIN Employee ON Department.Department_Id = Employee.Department_Id
WHERE Employee.Employee_Id IS NULL;
+---------------+--------+
| Department_Id | Name |
+---------------+--------+
|           201 | IT   |
+---------------+--------+
1 rows in set (0.00 sec);
```

12) List out the department wise maximum salary, minimum salary, average salary of the employees

```
mysql> SELECT Department_Id, MAX(Salary) AS Max_Salary, MIN(Salary) AS
Min_Salary, AVG(Salary) AS Avg_Salary FROM Employee GROUP BY
Department_Id;
+---------------+----------------+----------------+-------------------+
| Department_Id | Max_Salary | Min_Salary | Avg_Salary    |
+---------------+----------------+----------------+-------------------+
|           202 |   55000.00 |   44000.00 | 49500.000000 |
|           203 |   80000.00 |   80000.00 | 80000.000000 |
|           204 |   85000.00 |   85000.00 | 85000.000000 |
|           205 |   45000.00 |   45000.00 | 45000.000000 |
|           206 |   60000.00 |   60000.00 | 60000.000000 |
|           207 |   65000.00 |   55000.00 | 60000.000000 |
|           208 |   70000.00 |   70000.00 | 70000.000000 |
+---------------+----------------+----------------+-------------------+
7 rows in set (0.00 sec);
```

13) How many employees who are joined in 2022

```
mysql> SELECT COUNT(*) FROM Employee WHERE YEAR(Hiredate) = 2022;
+-------------+
| COUNT(*) |
+-------------+
|         9 |
+-------------+
1 rows in set (0.00 sec);
```

14) Display the employees who are working in "'South'"

```
mysql> SELECT E.Employee_Id, E.Lastname, E.Firstname FROM Employee E
JOIN Department D ON E.Department_Id = D.Department_Id JOIN Location L
ON D.Location_Id = L.Location_Id WHERE L.Reginal_Group = 'South';
+-----------------+-------------+--------------+
| Employee_Id | Lastname | Firstname |
+-----------------+-------------+--------------+
|         401 | M S      | Vishnu    |
|         402 | R S      | Lijith    |
|         405 | T        | Jithin    |
+-----------------+-------------+--------------+
3 rows in set (0.00 sec);
```

15) List our employees with their department names

```
mysql> SELECT e.Employee_Id, e.Lastname, e.Firstname, d.Name AS
Department_Name FROM Employee e JOIN Department d ON e.Department_Id =
d.Department_Id;
+-----------------+-----------------+--------------+-----------------------+
| Employee_Id | Lastname    | Firstname | Department_Name |
+-----------------+-----------------+--------------+-----------------------+
|         401 | M S         | Vishnu    | HR                |
|         402 | R S         | Lijith    | HR                |
|         403 | Simon       | Sam       | Operations        |
|         404 | Joby        | Aljo      | Research          |
|         405 | T           | Jithin    | Sales             |
|         406 | Davis       | Tessa     | Research          |
|         407 | Miller      | David     | Production        |
|         409 | Manadhavady | Teena     | Finance           |
|         410 | Kharim      | Ashik     | Marketing         |
+-----------------+-----------------+--------------+-----------------------+
10 rows in set (0.00 sec);
```

Result: The program is executed successfully and the output is obtained.

Experiment no: 6

View and Indexing in SQL

Aim: Execute DDL statements which demonstrate the use of views and Indexing. Try to update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables.

Objective: • To study and implement views and Indexing in DDL.

Theory : In MySQL, View is a virtual table created by a query by joining one or more tables.

MySQL Create VIEW

A VIEW is created by SELECT statements. SELECT statements are used to take data from the source table to make a VIEW.

Syntax:

```
CREATE [OR REPLACE] VIEW view_name AS
SELECT columns
FROM tables
[WHERE conditions];
```

Parameters:

OR REPLACE: It is optional. It is used when a VIEW already exist. If you do not specify this clause and the VIEW already exists, the CREATE VIEW statement will return an error.

view_name: It specifies the name of the VIEW that you want to create in MySQL.

WHERE conditions: It is also optional. It specifies the conditions that must be met for the records to be included in the VIEW.

Example:

The following example will create a VIEW name "trainer". This is a virtual table made by taking data from the table "courses".

```
CREATE VIEW trainer AS
SELECT course_name, course_trainer
FROM courses;
```

To see the created VIEW:

Syntax:

```
SELECT * FROM view_name;
```

Let's see how it looks the created VIEW:

```
SELECT * FROM trainer;
```

MySQL Update VIEW

In MYSQL, the ALTER VIEW statement is used to modify or update the already created VIEW without dropping it. **Syntax:**

```
ALTER VIEW view_name AS
SELECT columns
FROM table
WHERE conditions;
```

Example: The following example will alter the already created VIEW name "trainer" by adding a new column.

```
ALTER VIEW trainer AS
SELECT course_name, course_trainer, course_id FROM courses;
```

To see the altered VIEW:

```
SELECT*FROM trainer;
```

MySQL Drop VIEW

You can drop the VIEW by using the DROP VIEW statement. **Syntax:**

```
DROP VIEW [IF EXISTS] view_name; Parameters:
```

view_name: It specifies the name of the VIEW that you want to drop.

IF EXISTS: It is optional. If you do not specify this clause and the VIEW doesn't exist, the DROP VIEW statement will return an error.

Example:

```
DROP VIEW trainer;
```

Index

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

CREATE INDEX index_name ON table_name (column1, column2, ...);

Example:

CREATE INDEX idx_lastname ON Persons (LastName);[Simple Indexing]

CREATE INDEX idx_pname ON Persons (LastName, FirstName);[Composite Indexing]

DROP INDEX Statement

ALTER TABLE *table_name* DROP INDEX *index_name*;

OUTPUT

- mysql> create view location_view as select regional_group from location;
  Query OK, 0 rows affected (0.71 sec)

- mysql> select * from location_view;

```
+--------------------------+
| regional_group           |
+--------------------------+
| thiruvananthapuram       |
| kollam                   |
| pune                     |
| new york                 |
| mumbai                   |
| chennai                  |
| delhi                    |
| maharashtra              |
+--------------------------+
8 rows in set (0.10 sec)
```

- mysql> create index new_index on location(location_id);
  Query OK, 0 rows affected (0.87 sec)
  Records: 0 Duplicates: 0 Warnings: 0

- mysql> show indexes from location;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
location	0	PRIMARY	1	location_id	A	8	NULL	NULL		BTREE			YES	NULL
location	1	location_index	1	location_id	A	8	NULL	NULL		BTREE			YES	NULL
location	1	location_index	2	regional_group	A	8	NULL	NULL	YES	BTREE			YES	NULL
location	1	new_index	1	location_id	A	8	NULL	NULL		BTREE			YES	NULL

- mysql> alter table location drop index new_index;
  Query OK, 0 rows affected (0.28 sec)
  Records: 0 Duplicates: 0 Warnings: 0

- mysql> show indexes from location;

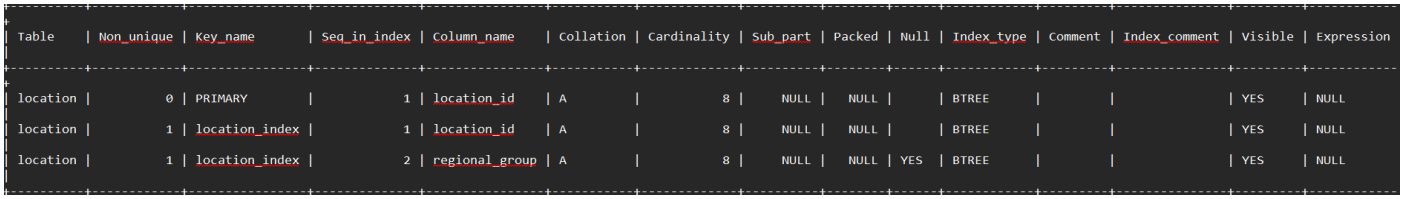


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
location	0	PRIMARY	1	location_id	A	8	NULL	NULL		BTREE			YES	NULL
location	1	location_index	1	location_id	A	8	NULL	NULL		BTREE			YES	NULL
location	1	location_index	2	regional_group	A	8	NULL	NULL	YES	BTREE			YES	NULL

Result: The program is executed successfully and the output is obtained.

Experiment no: 7

PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is Oracle Corporation's procedural extension for SQL and the Oracle relational database.PL/SQL includes procedural language elements such as conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variables of those types, and triggers. It can handle exceptions (runtime errors). One can create PL/SQL units such as procedures, functions, packages, types, and triggers, which are stored in the database for reuse by applications that use any of the Oracle Database programmatic interfaces.

Basics of PL/SQL

- PL/SQL stands for Procedural Language extensions to the Structured Query Language (SQL).
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- Oracle uses a PL/SQL engine to processes the PL/SQL statements.
- PL/SQL includes procedural language elements like conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variable of those types and triggers.

Disadvantages of SQL:

- SQL doesn't provide the programmers with a technique of condition checking, looping and branching.
- SQL statements are passed to Oracle engine one at a time which increases traffic and decreases speed.
- SQL has no facility of error checking during manipulation of data.

Features of PL/SQL:

- PL/SQL is basically a procedural language, which provides the functionality of decision making, iteration and many more features of procedural programming languages.
- PL/SQL can execute a number of queries in one block using single command.
- One can create a PL/SQL unit such as procedures, functions, packages, triggers, and types, which are stored in the database for reuse by applications.
- PL/SQL provides a feature to handle the exception which occurs in PL/SQL block known as exception handling block.
- Applications written in PL/SQL are portable to computer hardware or operating system where Oracle is operational.
- PL/SQL Offers extensive error checking.

Differences between SQL and PL/SQL:

SQL	PL/SQL
SQL is a single query that is used to perform DML and DDL operations.	PL/SQL is a block of codes that used to write the entire program blocks/ procedure/ function, etc.
It is declarative, that defines what needs to be done, rather than how things need to be done.	PL/SQL is procedural that defines how the things needs to be done.
Execute as a single statement.	Execute as a whole block.
Mainly used to manipulate data.	Mainly used to create an application.
Cannot contain PL/SQL code in it.	It is an extension of SQL, so it can contain SQL inside it.

Structure of PL/SQL Block:

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.

Typically, each block performs a logical action in the program. A block has the following structure:

DECLARE
declaration statements;
BEGIN executable statements

EXCEPTIONS
exception handling statements

END

Experiment no: 8

Volume of Cuboid

- ```
  SQL> declare
  l integer;
  b integer;
  h integer;
  vol integer;
  begin
  l:=&l;
  b:=&b;
  h:=&h;
  vol:=l*b*h;
  dbms_output.put_line(volume of cuboid: '||vol);
  end;
  /
  ```

- **Output**

```
Enter value for l: 3
old   7: l:=&l;
new   7: l:=3;
Enter value for b: 4
old   8: b:=&b;
new   8: b:=4;
Enter value for h: 5
old   9: h:=&h;
new   9: h:=5;
volume of cuboid: 60
PL/SQL procedure successfully completed
```

Result: The program executed successfully and output is obtained.
```

Experiment no: 9

Largest of Three Numbers

- SQl>DECLARE
```
    num1 NUMBER;
    num2 NUMBER;
    num3 NUMBER;
    largest NUMBER;
   BEGIN
    num1 := &Enter_First_Number;
    num2 := &Enter_Second_Number;
    num3 := &Enter_Third_Number;
    IF num1 >= num2 AND num1 >= num3 THEN
        largest := num1;
    ELSIF num2 >= num1 AND num2 >= num3 THEN
        largest := num2;
    ELSE
        largest := num3;
    END IF;
    DBMS_OUTPUT.PUT_LINE('The largest number is: ' || largest);
END;
/
```
- **output**

```
Enter value for num1: 25
old   7:     num1 := &num1;
new   7:     num1 := 25
Enter value for num2: 42
old   8:     num2 := &num2;
new   8:     num2 := 42
Enter value for num3: 17
old   9:     num3 := &num3;
new   9:     num3 := 17
The largest number is: 42

PL/SQL procedure successfully completed.
```

Result: The program executed successfully and output is obtained.

Experiment no: 10

Factorial of a Number

- ```
  SQL> declare
  f number:=1;
  n number;
  i number;
  begin
  n:=&n;
  for i in 1..n
  loop
   f:=f*i;
  end loop;
  dbms_output.put_line('factorial is: '||f);
  end;
  /
  ```

- **output**

  ```
  Enter value for n: 5
  old 6: n:=&n;
  new 6: n:=5;
  factorial is: 120
  PL/SQL procedure successfully completed
  ```

Result: The program executed successfully and output is obtained.
  ```

Experiment no: 11

Sum of digits of a Number

- ```
  SQL> declare
  s number:=0;
  n integer;
  rem integer;
  begin
  n:=&n;
  while n>0
  loop
  rem:=mod(n,10);
  s:=s+rem;
  n:=trunc(n/10);
  end loop;
  dbms_output.put_line('sum of digit is:'||s);
  end;
  /
  ```

- **output**

  ```
  Enter value for n: 123
  old 6: n:=&n;
  new 6: n:=123;
  sum of digit is:6
  PL/SQL procedure successfully completed
  ```

Result: The program executed successfully and output is obtained.
  ```

Experiment no: 12

Sum of n numbers

- ```
  SQL> declare
  n  integer;
  s number:=0;
  begin
  n:=&n;
  for i in 1..n
  loop
  s:=s+i;
  end loop;
  dbms_output.put_line('Sum is:'||s);
  end;
  /
  ```

- **Output**

  ```
  Enter value for n:5
  old   5: n:=&n;
  new   5: n:=5;
  Sum is:15
  PL/SQL procedure successfully completed
  ```

Result: The program executed successfully and output is obtained.
  ```

Experiment no: 13

Reverse of a string

- ```
  SQL> declare
  str varchar(20):='&str';
  len integer;
  str1 varchar(20);
  begin
  len:=length(str);
  for i in reverse 1..len loop
  str1:=str1||substr(str,i,1);
  end loop;
  dbms_output.put_line('reverse is:'||str1);
  end;
  /
  ```

- **Output**

  ```
  Enter value for str: nice
  old  2:  str  varchar(20):='&str';
  new  2:  str  varchar(20):='nice';
  reverse is:ecin
  PL/SQL procedure successfully completed
  ```

Result: The program executed successfully and output is obtained.
  ```

Experiment no: 14

Reverse of a number

- ```
  SQL> declare
  rev integer:=0;
  rem integer;
  n integer;
  begin
  n:=&n;
  while n>0
  loop
  rem:=mod(n,10);
  rev:=rev*10+rem;
  n:=trunc(n/10);
  end loop;
  dbms_output.put_line('the reverse is'||rev);
  end;
  /
  ```

- **Output**

  ```
  Enter value for n:123
  old   6: n:=&n;
  new   6: n:=123;
  the reverse is321
  PL/SQL procedure successfully completed
  ```

Result: The program executed successfully and output is obtained.
  ```

Experiment no: 15

Palindrome or not

- ```
  SQL> declare
  n integer;
  rev integer:=0;
  rem integer;
  temp integer;
  begin
  n:=&n;
  temp:=n;
  while n>0
  loop
  rem:=mod(n,10);
  rev:=rev*10+rem;
  n:=trunc(n/10);
  end loop;
  if temp=rev
  then
  dbms_output.put_line('palindrome');
  else
  dbms_output.put_line('not palindrome');
  end if;
  end;
  /
  ```

- **Output**

  ```
  Enter value for n: 121
  old   7: n:=&n;
  new   7: n:=121;
  palindrome
  PL/SQL procedure successfully completed
  ```

Result: The program executed successfully and output is obtained.
  ```

Experiment no: 16

Armstrong Number or Not

- ```
  DECLARE
      n     NUMBER;
      temp  NUMBER;
      digit NUMBER;
      sum   NUMBER := 0;
      len   NUMBER;
  BEGIN
      n := &n;
      temp := n;
      len := LENGTH(TO_CHAR(n));

      WHILE temp > 0 LOOP
          digit := MOD(temp, 10);
          sum := sum + POWER(digit, len);
          temp := TRUNC(temp / 10);
      END LOOP;

      IF sum = n THEN
          DBMS_OUTPUT.PUT_LINE('The given number ' || n || ' is an Armstrong
     number');
      ELSE
          DBMS_OUTPUT.PUT_LINE('The given number ' || n || ' is not an Armstrong
     number');
      END IF;
  END;
  /
  ```

- **Output**

  ```
  Enter value for n: 153
  old   8:     n := &n;
  new   8:     n := 153
  The given number 153 is an Armstrong number

  PL/SQL procedure successfully completed.
  ```

Result: The program executed successfully and output is obtained.
  ```

Experiment no: 17

Cursor

Aim: Write a PL/SQL block to implement all types of cursor.

Objective:

- To study and implement PL/SQL cursors.

Theory :

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it. Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected. In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK_ROWCOUNT** and **%BULK_EXCEPTIONS**, designed for Use with the **FORALL** statement.

The following table provides the description of the most used attributes −

S.No	Attribute & Description
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area.** An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT statement which returns more than one row.

The syntax for creating an explicit cursor is –

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above- opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Experiment no: 18

IMPLICIT CURSOR

- SQL> create table customerr(cust_id varchar(5),name varchar(10),salary int);
  Table created.

- SQL> insert into customerr values('co1','david',23000);
  1 row created.

- SQL> insert into customerr values('co2','bob',12000);
  1 row created.

- SQL> insert into customerr values('c03','lilly',24000);
  1 row created.

- SQL> insert into customerr values('co4','shwetha',56000);
  1 row created.

- SQL> select * from customerr;

CUST_ID	NAME	SALARY
co1	david	23000
co2	bob	12000
c03	lilly	24000
co4	shwetha	56000

- ```
  declare
  total_rows number(2);
  begin
  update customerr set salary=salary+5000;
  if sql%notfound
  then
  dbms_output.put_line('No customers selected');
  elsif sql%found then
  total_rows:=sql%rowcount;
  dbms_output.put_line(total_rows||'customers selected');
  end if;
  end;
  /
  ```

- SQL> @implicit.sql
  4customers selected
  PL/SQL procedure successfully completed.
  ```

- SQL> select*from customerr;

```
CUST_ID NAME        SALARY
---------- ---------- ----------
co1     david       28000
co2     bob         17000
c03     lilly       29000
co4     shwetha     61000
```

Result: The program is executed successfully and the output is obtained.

Experiment no: 19

EXPLICIT CURSOR 1

- SQL> create table customerr(cust_id varchar(5),name varchar(10),salary int);
  Table created.

- SQL> insert into customerr values('co1','david',28000);
  1 row created.

- SQL> insert into customerr values('co2','bob',17000);
  1 row created.

- SQL> insert into customerr values('c03','lilly',29000);
  1 row created.

- SQL> insert into customerr values('co4','shwetha',61000);
  1 row created.

- SQL> declare
  c_id customerr.cust_id%type;
  c_name customerr.name%type;
  c_salary customerr.salary%type;
  cursor c1 is select cust_id,name,salary from customerr;
  begin
  open c1;
  loop
  fetch c1 into c_id,c_name,c_salary;
  exit when c1%notfound;
  dbms_output.put_line(c_id||' '||c_name||' '||c_salary);
  end loop;
  close c1;
  end;
  /

- SQL> @explicit.sql
  co1 david 28000
  co2 bob 17000
  c03 lilly 29000
  co4 shwetha 61000
  PL/SQL procedure successfully completed.

Result: The program is executed successfully and the output is obtained.

Experiment no: 20

EXPLICIT CURSOR 2

- ```
  SQL> declare
  c_id customer.custid%type;
  c_name customer.custname%type;
  c_phno customer.phno%type;
  cursor c1 is select custid,custname,phno from customer;
  begin
  open c1;
  loop
  fetch c1 into c_id,c_name,c_phno;
   exit when c1%notfound;
  dbms_output.put_line(c_id||' '||c_name||' '||c_phno);
  if c_id=1 then
  c_phno:=c_phno+1000;
  elsif c_id=2 then
  c_phno:=c_phno+2000;
  end if;
  update customer set phno=c_phno where custid=c_id;
  end loop;
  close c1;
  end;
  /
  ```

Result: The program is executed successfully and the output is obtained.
  ```

Experiment no: s21

Trigger

Aim: Write and execute suitable database triggers .Consider row level and statement level triggers.

Objective:

- To study and implement PL/SQL triggers.

Theory :

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events.

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes −

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is –

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
```

```
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers

Experiment no: 22

Trigger 1

- cec@user:~$ sudo su
  [sudo] password for cec:
  root@user:/home/cec# sqlplus sys as sysdba

- SQL*Plus: Release 11.2.0.2.0 Production on Mon Apr 22 14:38:16 2024
  Copyright (c) 1982, 2011, Oracle. All rights reserved.
  Enter password:
  Connected to:
  Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production

- SQL> create user u2 identified by abc123;
  User created.
- SQL> grant connect,resource to u2;
  Grant succeeded.

- SQL> exit;
  Disconnected from Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
  root@user:/home/cec# sqlplus u2 as sysdba;

- SQLPlus: Release 11.2.0.2.0 Production on Mon Apr 22 14:40:20 2024
  Copyright (c) 1982, 2011, Oracle. All rights reserved.
  Enter user-name: u2
  Enter password:
  Connected to:
  Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production

- SQL> create table customer(id integer,name varchar(10),salary integer);
  Table created.

- SQL> set serveroutput on
  create or replace trigger t1 before insert or delete or update on customer for each row
  when(new.id>0)
  declare
  sal_diff number;
  begin
  sal_diff:=:new.salary-:old.salary;
  dbms_output.put_line('old salary'||:old.salary);
  dbms_output.put_line('new salary'||:new.salary);
  dbms_output.put_line('salary diff'||sal_diff);
  end;
  /

- ```
  SQL> @triggerr.sql
  Trigger created.
  ```

- ```
  SQL> insert into customer values(01,'joe',5000);
  old salary
  new salary5000
  salary diff
  1 row created.
  ```

- ```
  SQL> insert into customer values(02,'dia',2300);
  old salary
  new salary2300
  salary diff
  1 row created.
  ```

- ```
  SQL> insert into customer values(03,'lilly',9000);
  old salary
  new salary9000
  salary diff
  1 row created.
  ```

- ```
  SQL> update customer set salary=45000 where id=02;
  old salary2300
  new salary45000
  salary diff42700
  1 row updated.
  ```

Result: The program is executed successfully and the output is obtained.
  ```

Experiment no: 23

Trigger 2

- SQL> create table stock(item varchar(10),stock_avail int);
  Table created.

- SQL> create table purchase(id int,item varchar(10),no_of_items int,price int);
  Table created.

- SQL> insert into stock values('pen',100);
  1 row created.

- SQL> insert into stock values('pencil',200);
  1 row created.

- SQL> select * from stock;

```
ITEM        STOCK_AVAIL
---------- -----------
pen          100
pencil       200
```

- SQL> desc purchase;

```
 Name                              Null?    Type
 ----------------------------------- -------- ------------------------
 ID                                          NUMBER(38)
 ITEM                                        VARCHAR2(10)
 NO_OF_ITEMS                                 NUMBER(38)
 PRICE                                       NUMBER(38)
```

- SQL> create or replace trigger stock_updation after insert on purchase
  for each row
  when(new.no_of_item>0)
  begin
  update stock set stockavail=stockavail-:new.no_of_item where item=:new.item;
  dbms_output.put_line('stock update');
  end;
  /

- SQL> @triggerr2.sql
  Trigger created.

- SQL> insert into purchase values(01,'pencil',10,12);
  stock updated
  1 row created.

- ```
  SQL> insert into purchase values(02,'eraser',20,5);
  stock updated
  1 row created.
  ```

- ```
  SQL> insert into purchase values(03,'pen',100,20);
  stock updated
  1 row created.
  ```

- ```
  SQL> select * from purchase;
  ID          ITEM        NO_OF_ITEMS  PRICE
  ----------- ----------- ------------ -----------
  1           pencil      10            12
  2           eraser      20             5
  3           pen         100            20
  ```

- ```
  SQL> select * from stock;
  ITEM        STOCK_AVAIL
  ----------- ---------------
  pen          0
  pencil       190
  ```

Result: The program is executed successfully and the output is obtained.

Experiment no: 24

MongoDB Installation

Install and configure client and server for MongoDB (Show all commands and necessary steps for installation and configuration).

MongoDB

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data. The following table shows the relationship of RDBMS terminology with MongoDB.

MongoDB Features

- **General purpose database**, almost as fast as the key:value NoSQL type.
- **High availability**.
- **Scalability** (from a standalone server to distributed architectures of huge clusters). This allows us to shard our database transparently across all our shards. This increases the performance of our data processing.
- **Aggregation**: batch data processing and aggregate calculations using native MongoDB operations.
- **Load Balancing**: automatic data movement across different shards for load balancing. The balancer decides when to migrate the data and the destination Shard, so they are evenly distributed among all servers in the cluster. Each shard stores the data for a selected range of our collection according to a partition key.
- **Native Replication**: syncing data across all the servers at the replica set.
- **Security**: authentication, authorization, etc.
- **Advanced users management**.
- **Automatic failover**: automatic election of a new primary when it has gone down.

Installation Steps for MongoDB

1. Connect the system with the Internet.
2. Open the terminal and Execute the command sudo apt-get update sudo apt-get install mongodb sudo service mongodb start
3. Type mongo to start the mongodb terminal.
4. Now write your queries.

Result : The installation of MongoDB is done successfully.

Experiment no: 25

MongoDB CRUD Operations

MongoDB

MongoDB is an open-source document database and leading NoSQL database.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

Advantages of MongoDB over RDBMS

- **Schema-less:** MongoDB is a document database where a single collection can hold different documents. The number of fields, content, and size of each document can vary from one to another.
- **Clear Structure:** The structure of a single object is straightforward, eliminating the need for complex joins.
- **Deep Query-ability:** MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- **Tuning**: MongoDB offers various tuning options to optimize performance based on application requirements.
- **Ease of Scale-Out:** MongoDB is designed for easy scalability, allowing for horizontal scaling across multiple servers.
- **No Object-Relational Mapping Needed:** Conversion or mapping of application objects to database objects is not required, simplifying development.
- **In-Memory Storage:** MongoDB uses internal memory for storing the (windowed) working set, enabling faster access to data.

Why Use MongoDB?

- Document-Oriented Storage: Data is stored in the form of JSON-style documents.
- Index on Any Attribute
- Replication and High Availability
- Auto-Sharding
- Rich Queries
- Fast In-Place Updates
- Professional Support by MongoDB

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

The use Command

MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

```
use DATABASE_NAME
```

Example

```
>use mydb
switched to
```

To check your currently selected database, use the command **db**

```
>db
mydb
```

If you want to check your databases list, use the command **show dbs**.

```
>show dbs
local
        0.78125GB
```

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop a existing database. **Syntax**

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

Example

First, check the list of available databases by using the command, **show dbs**.

```
>show dbs
local0.78125GB
mydb0.23012GB
test0.23012GB
>
```

If you want to delete new database **<mydb>**, then **dropDatabase()**command would be as follows −

```
>use mydb
switched
to dbmydb
>db.dropDatabase()
```

```
>show dbs
local0.78125GB
test0.23012GB
>
```

The createCollection() Method

MongoDB **db.createCollection(name, options)** is used to create collection. **Syntax**

```
db.createCollection(name, options)
```

Examples

```
>use test switched
to db test
>db.createCollection("mycollection")
{"ok":1}
>
```

You can check the created collection by using the command **show collections**.

```
>show
collections
mycollection
```

The drop() Method

MongoDB's **db.collection.drop()** is used to drop a collection from the database.

Syntax

Basic syntax of **drop()** command is as follows −

```
db.COLLECTION_NAME.drop()
```

The insert() Method

To insert data into a MongoDB collection, you can use MongoDB's insert() or save() method.

Syntax

The basic syntax of the insert() command is as follows:

```
db.COLLECTION_NAME.insert(document)
```

Example

```
>db.mycol.insert({
   _id:ObjectId(7df78ad8902c),
title:'MongoDB Overview',
description:'MongoDB is no sql database', by:'tutorials point',
url:'http://www.tutorialspoint.com', tags: ['mongodb','database','NoSQL'], likes:100 })
```

Here **mycol** is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.

_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows –

```
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id,
   3 bytes incrementer)
```

To insert multiple documents in a single query, you can pass an array of documents in insert() command. Example

```
> db.post.insert([
  {
    title: 'MongoDB Overview',
    description: 'MongoDB is a NoSQL database',
    by: 'TutorialsPoint',
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
  },
  {
    title: 'NoSQL Database',
    description: "NoSQL databases don't have tables",
    by: 'TutorialsPoint',
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 20,
    comments: [
      {
        user: 'user1',
        message: 'My first comment',
        dateCreated: new Date(2013, 11, 10, 2, 35),
        like: 0
      }
    ]
  }
])
```

The find() Method

To query data from MongoDB collection, you need to use MongoDB's **find()**method.

Syntax

The basic syntax of **find()** method is as follows −

```
>db.COLLECTION_NAME.find()
```

find() method will display all the documents in a non-structured way.

The pretty() Method

To display the results in a formatted way, you can use **pretty()** method. Syntax

```
>db.mycol.find().pretty()
```

MongoDBUpdate() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows −

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA,  UPDATED_DATA)
```

Example

Consider the mycol collection has the following data.

```
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'NewMongoDB Tutorial'}})
>db.mycol.find()
{"_id":ObjectId(5983548781331adf45ec5),"title":"NewMongoDB Tutorial"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
>
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'},
{$set:{'title':'NewMongoDB Tutorial'}},{multi:true})
```

MongoDBSave() Method

The **save()** method replaces the existing document with the new document passed in the save() method.

Syntax

The basic syntax of MongoDB **save()** method is shown below −

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

deletion criteria − (Optional) deletion criteria according to documents will be removed.

justOne − (Optional) if set to true or 1, then remove only one document.

Syntax

Basic syntax of **remove()** method is as follows −

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Example

Consider the mycol collection has the following data.

```
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

Following example will remove all the documents whose title is 'MongoDB Overview'.

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
>
```

Remove Only One

If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove()** method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

```
>db.mycol.remove()
>db.mycol.find()
>
```

The find() Method

Execute at least 10 queries on any suitable MongoDB database that demonstrates following:

$ where queries

Cursors (Limits, skips, sorts, advanced query options)

Database commands

MongoDB's **find()** method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute **find()** method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

Syntax

The basic syntax of **find()** method with projection is as follows −

```
>db.COLLECTION_NAME.find({},{KEY:1})
```

Following example will display the title of the document while querying the document.

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
>
```

Apart from find() method there is findOne() method, that reruns only one document.

AND in MongoDB

Syntax

In the **find()** method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND** condition. Following is the basic syntax of **AND**

```
>db.mycol.find(
   {
      $and: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```
>db.mycol.find({$and:[{"by":"tutorials point"},{"title":"MongoDB Overview"}]}).pretty(){
"_id":ObjectId(7df78ad8902c),
"title":"MongoDB Overview",
"description":"MongoDB is no sql database",
"by":"tutorials point",
"url":"http://www.tutorialspoint.com",
"tags":["mongodb","database","NoSQL"],
"likes":"100"
}
```

For the above given example, equivalent where clause will be ' where by = 'tutorials point' AND title = 'MongoDB Overview' '. You can pass any number of key, value pairs in find clause.

OR in MongoDB

Syntax

To query documents based on the OR condition, you need to use **$or**keyword. Following is the basic syntax of **OR** −

```
>db.mycol.find(
{
    $or:[
{key1: value1}
,{key2:value2}
]
}
).pretty()
```

Example

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

```
>db.mycol.find({$or:[{"by":"tutorials point"},{"title":"MongoDB Overview"}]}).pretty()
{
"_id":ObjectId(7df78ad8902c),
"title":"MongoDB Overview",
```

```
"description":"MongoDB is no sql database",
"by":"tutorials point",
"url":"http://www.tutorialspoint.com",
"tags":["mongodb","database","NoSQL"],
"likes":"100"
}
>
```

Using AND and OR Together

Example

The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is **'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

```
>db.mycol.find({"likes":{$gt:10}, $or:[{"by":"tutorials point"},
{"title":"MongoDB Overview"}]}).pretty()
```

The Limit() Method

To limit the records in MongoDB, you need to use **limit()** method. The method accepts one number type argument, which is the number of documents that you want to be displayed. **Syntax**

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

Example

Consider the collection myycol has the following data.

```
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty( )	where likes < 50
Less Than Equals	{<key>:{$lte:<value>} }	db.mycol.find({"likes":{$lte:50}}).prett y()	where likes <= 50

Greater Than	{<key>:{$gt:<value>} }	db.mycol.find({"likes":{$gt:50}}).pretty ()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value> }}	db.mycol.find({"likes":{$gte:50}}).prett y()	where likes >= 50
Not Equals	{<key>:{$ne:<value>} }	db.mycol.find({"likes":{$ne:50}}).pretty ()	where likes != 50

OUTPUT

```
> use abhishek
  switched to db abhishek

> db; - will display the curreent database.
  abhishek

> db.createCollection("Student"); - to create a Student collection( In
  MONGOdb collection is a Table)
  { "ok" : 1 }

> show collections; - to check the collection.
  Student

> db.Student.insert({"Roll_no":1,"Name":"Anu"}); - to insert within a
  database.
  WriteResult({ "nInserted" : 1 })
> db.Student.insert({"Roll_no":2,"Name":"Abhi"});
  WriteResult({ "nInserted" : 1 })
> db.Student.insert({"Roll_no":3,"Name":"Shwetha"});
  WriteResult({ "nInserted" : 1 })
> db.Student.insert({"Roll_no":4,"Name":"Nayan"});
  WriteResult({ "nInserted" : 1 })
> db.Student.insert({"Roll_no":5,"Name":"Sreejith"});
  WriteResult({ "nInserted" : 1 })

INSERTED WITHIN Student database.

> db.Student.find() - to dispaly the inserted.
  { "_id" : ObjectId("6627797c96400503b4c15967"), "Roll_no" : 1, "Name" :
  "Anu" }
  { "_id" : ObjectId("662779b196400503b4c15969"), "Roll_no" : 2, "Name" :
  "Abhi" }
  { "_id" : ObjectId("662779bb96400503b4c1596a"), "Roll_no" : 3, "Name" :
  "Shwetha" }
  { "_id" : ObjectId("662779c896400503b4c1596b"), "Roll_no" : 4, "Name" :
  "Nayan" }
  { "_id" : ObjectId("662779d396400503b4c1596c"), "Roll_no" : 5, "Name" :
  "Sreejith" }
```

```
> db.Student.find().pretty() - Configures the cursor to display results in
   a format that is easy to read.
  {
      "_id" : ObjectId("6627797c96400503b4c15967"),
      "Roll_no" : 1,
      "Name" : "Anu"
  }
  {
      "_id" : ObjectId("662779b196400503b4c15969"),
      "Roll_no" : 2,
      "Name" : "Abhi"
  }
  {
      "_id" : ObjectId("662779bb96400503b4c1596a"),
      "Roll_no" : 3,
      "Name" : "Shwetha"
  }
  {
      "_id" : ObjectId("662779c896400503b4c1596b"),
      "Roll_no" : 4,
      "Name" : "Nayan"
  }
  {
      "_id" : ObjectId("662779d396400503b4c1596c"),
      "Roll_no" : 5,
      "Name" : "Sreejith"
  }


> db.Student.findOne() - will display the first inserted data according to
  a sequence.
  {
      "_id" : ObjectId("6627797c96400503b4c15967"),
      "Roll_no" : 1,
      "Name" : "Anu"
  }

> db.Student.find().limit(3); - will display first three from the database.
  { "_id" : ObjectId("6627797c96400503b4c15967"), "Roll_no" : 1, "Name" :
  "Anu" }
  { "_id" : ObjectId("6627799296400503b4c15968"), "Roll_no" : 2, "Name" :
  "Abhi" }
```

```
  { "_id" : ObjectId("662779b196400503b4c15969"), "Roll_no" : 3, "Name" :
  "Shwetha" }

> db.Student.find().limit(3).skip(2); - will display after the first two
  are deleted.
  { "_id" : ObjectId("662779b196400503b4c15969"), "Roll_no" : 3, "Name" :
  "Shwetha" }
  { "_id" : ObjectId("662779bb96400503b4c1596a"), "Roll_no" : 4, "Name" :
  "Nayan" }
  { "_id" : ObjectId("662779c896400503b4c1596b"), "Roll_no" : 5, "Name" :
  "Sreejith" }


> db.Student.update({"Roll_no":2},{$set:{"Name":"goutham"}}); - to
  change/update the name or any data.
  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Student.find();
  { "_id" : ObjectId("6627797c96400503b4c15967"), "Roll_no" : 1, "Name" :
  "Anu" }
  { "_id" : ObjectId("6627799296400503b4c15968"), "Roll_no" : 2, "Name" :
  "goutham" }
  { "_id" : ObjectId("662779b196400503b4c15969"), "Roll_no" : 3, "Name" :
  "Shwetha" }
  { "_id" : ObjectId("662779c896400503b4c1596b"), "Roll_no" : 4, "Name" :
  "Nayan" }
  { "_id" : ObjectId("662779d396400503b4c1596c"), "Roll_no" : 5, "Name" :
  "Sreejith" }

> db.Student.remove({"Roll_no":3}); - to remove a row.
  WriteResult({ "nRemoved" : 1 })
> db.Student.find();
  { "_id" : ObjectId("6627799296400503b4c15968"), "Roll_no" : 1, "Name" :
  "Anu" }
  { "_id" : ObjectId("662779b196400503b4c15969"), "Roll_no" : 2, "Name" : "
  goutham" }
  { "_id" : ObjectId("662779c896400503b4c1596b"), "Roll_no" : 4, "Name" :
  "Nayan" }
  { "_id" : ObjectId("662779d396400503b4c1596c"), "Roll_no" : 5, "Name" :
  "Sreejith" }


> db.Student.find().sort({"Name":1}); - will sort in ascending order.
```

```
  { "_id" : ObjectId("6627799296400503b4c15968"), "Roll_no" : 1, "Name" :
  "Anu" }
  { "_id" : ObjectId("662779c896400503b4c1596b"), "Roll_no" : 4, "Name" :
  "Nayan" }
  { "_id" : ObjectId("662779d396400503b4c1596c"), "Roll_no" : 5, "Name" :
  "Sreejith" }
  { "_id" : ObjectId("662779b196400503b4c15969"), "Roll_no" : 2, "Name" :
  "goutham" }
  { "_id" : ObjectId("6627797c96400503b4c15967"), "Roll_no" : 1, "Name" :
  "nahas" }

> db.Student.find().sort({"Name":-1}); -  will sort in descending order.
  { "_id" : ObjectId("6627797c96400503b4c15967"), "Roll_no" : 1, "Name" :
  "nahas" }
  { "_id" : ObjectId("662779b196400503b4c15969"), "Roll_no" : 2, "Name" :
  "goutham" }
  { "_id" : ObjectId("662779d396400503b4c1596c"), "Roll_no" : 5, "Name" :
  "Sreejith" }
  { "_id" : ObjectId("662779c896400503b4c1596b"), "Roll_no" : 4, "Name" :
  "Nayan" }
  { "_id" : ObjectId("6627799296400503b4c15968"), "Roll_no" : 1, "Name" :
  "Anu" }
```

Result: The program is executed successfully and the output is obtained.