Faculty Name/s**: Dr Prathibha G**			Academic Year:2023-2024				
Department: Information Science and Engineering							
Course Code	Course Title	Core/Elective	Prerequisite	Contact Hours			Total Hrs/ Sessions
				L	T	P	
BCS403	**Database Management System**	**Core**		**3**	**0**	**2**	**40**

PRACTICAL COMPONENT OF IPCC

1. Create a table called Employee & execute the following.

Employee(EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)

1. Create a user and grant all permissions to theuser.
2. Insert the any three records in the employee table contains attributes EMPNO,ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result.
3. Add primary key constraint and not null constraint to the employee table.
4. Insert null values to the employee table and verify the result.

2. Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following.

1. Add a column commission with domain to the Employeetable.
2. Insert any five records into the table.
3. Update the column details of job
4. Rename the column of Employ table using alter command.
5. Delete the employee whose Empno is 105.

3. Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by,Orderby.

Employee(E_id, E_name, Age, Salary)

1. Create Employee table containing all Records E_id, E_name, Age, Salary.
2. Count number of employee names from employeetable
3. Find the Maximum age from employee table.
4. Find the Minimum age from employeetable.
5. Find salaries of employee in Ascending Order.
6. Find grouped salaries of employees.

4. Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.
CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)
5. Create cursor for Employee table & extract the values from the table. Declare the variables ,Open the cursor & extrct the values from the cursor. Close the cursor.
Employee(E_id, E_name, Age, Salary)
6. Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.
7. Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.

Course Outcomes: Students will be able to

CO1: Describe the basic elements of a relational database management system.
CO2: Design entity relationship for the given scenario.
CO3: Apply various Structured Query Language (SQL) statements for database manipulation.
CO4: Analyse various normalization forms for the given application
CO5: Develop database applications for the given real world problem.
CO6: Understand the concepts related to NoSQL databases.

The Correlation of Course Outcomes (CO's) and Program Outcomes (PO's)

Subject Code:	**BCS403**		**TITLE: Database Management System**			**Faculty Name:**			**Dr.PRATHIBHA G**					
Course Outcomes	**Program Outcomes**													
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO 10**	**PO 11**	**PO 12**	**PSO 1**	**PSO 2**	
CO-1	3		2								3	3	3		
CO-2	2	2	3	3	3							2	2	3	
CO-3	2	2	3	3	3						2	2	2	3	
CO-4	2	2	3	3	3						3	3	3		
CO-5	2	2	3	3	3						3	3	2	3	
CO-6	3	3			3							3	3		

Note:3 = Strong Contribution 2 = Average Contribution 1 = Weak Contribution

Program 1

Create a table called Employee & execute the following.

Employee(EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)

1. Create a user and grant all permissions to theuser.

2. Insert the any three records in the employee table contains attributes

EMPNO,ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback.

Check the result.

3. Add primary key constraint and not null constraint to the employee table.

4. Insert null values to the employee table and verify the result.

Solution:

Lets login with the **root** account as shown below. Create a database **COMPANY** and switch to it using the **USE** command.

```
$ sudo mysql -u root

mysql> CREATE DATABASE COMPANY;
Query OK, 1 row affected (0.14 sec)

mysql> USE COMPANY;
Database changed
```

Creating the Employee Table

Within the Database **COMPANY** create a table **Employee** as follows. Use the **SHOW TABLES;** command to confirm that the table was indeed created.

```
mysql> CREATE TABLE COMPANY.Employee (
    ->     EMPNO INT,
    ->     ENAME VARCHAR(255),
    ->     JOB VARCHAR(255),
    ->     MANAGER_NO INT,
    ->     SAL DECIMAL(10, 2),
    ->     COMMISSION DECIMAL(10, 2)
    -> );

Query OK, 0 rows affected (0.91 sec)


mysql> SHOW TABLES;

+------------------+
| Tables_in_COMPANY |
+------------------+
| Employee         |
+------------------+
1 row in set (0.00 sec)
```

You can verify the structure of this newly created Employee table using the DESC command.

```
mysql> DESC COMPANY.Employee;

+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| EMPNO      | int           | YES  |     | NULL    |       |
| ENAME      | varchar(255)  | YES  |     | NULL    |       |
| JOB        | varchar(255)  | YES  |     | NULL    |       |
| MANAGER_NO | int           | YES  |     | NULL    |       |
| SAL        | decimal(10,2) | YES  |     | NULL    |       |
| COMMISSION | decimal(10,2) | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Create a User and Grant Permissions

```
mysql> CREATE USER IF NOT EXISTS 'dbuser'@'localhost' IDENTIFIED BY 'T0p5E(RET';

mysql> GRANT ALL PRIVILEGES ON COMPANY.Employee TO 'dbuser'@'localhost';
```

Now logout and login with the new account credentials. Press **Ctrl+D** to logout. Command to login with new user account is shown below.

```
$ mysql -u dbuser -p
Enter password:

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.37 MySQL Community Server - GPL

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>
```

Now you have successfully logged with your new account. Change the current database to **COMPANY** database using **USE** command. Now we will illustrate how to **insert** records and also the **COMMIT** and **ROLLBACK** facilities.

```
-- Change the current database to COMPANY

mysql> USE COMPANY;

Database changed
```

```
mysql> SELECT * FROM Employee;

Query OK, 0 rows affected (0.00 sec)

-- START A TRANSACTION
mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
    -> VALUES (1, 'Kavana Shetty', 'Manager', NULL, 5000.00, 1000.00);

Query OK, 1 row affected (0.00 sec)

-- COMMIT DATABASE, db CONTENTS ARE WRITTEN TO THE DISK
mysql> COMMIT;

Query OK, 0 rows affected (0.06 sec)

-- DISPLAY TABLE CONTENTS
mysql> SELECT * FROM Employee;
+-------+--------------+---------+------------+---------+------------+
| EMPNO | ENAME        | JOB     | MANAGER_NO | SAL     | COMMISSION |
+-------+--------------+---------+------------+---------+------------+
|     1 | Kavana Shetty | Manager |       NULL | 5000.00 |    1000.00 |
+-------+--------------+---------+------------+---------+------------+
1 row in set (0.00 sec)


-- START ANOTHER TRANSACTION
mysql> START TRANSACTION;

-- INSERT MORE RECORDS
mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (2, 'Ram Charan', 'Developer', 1, 4000.00, NULL);

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (3, 'Honey Singh', 'Salesperson', 2, 3000.00, 500.00);

mysql> SELECT * FROM Employee;
+-------+--------------+-------------+------------+---------+------------+
| EMPNO | ENAME        | JOB         | MANAGER_NO | SAL     | COMMISSION |
+-------+--------------+-------------+------------+---------+------------+
|     1 | Kavana Shetty | Manager     |       NULL | 5000.00 |    1000.00 |
|     2 | Ram Charan    | Developer   |          1 | 4000.00 |       NULL |
|     3 | Honey Singh   | Salesperson |          2 | 3000.00 |     500.00 |
+-------+--------------+-------------+------------+---------+------------+
3 rows in set (0.00 sec)

mysql> DELETE FROM Employee where ENAME = 'Kavana Shetty';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Employee;
+-------+-------------+-------------+------------+---------+------------+
| EMPNO | ENAME       | JOB         | MANAGER_NO | SAL     | COMMISSION |
+-------+-------------+-------------+------------+---------+------------+
|     2 | Ram Charan  | Developer   |          1 | 4000.00 |       NULL |
|     3 | Honey Singh | Salesperson |          2 | 3000.00 |     500.00 |
```

```
+--------+------------+------------+-----------+---------+------------+
2 rows in set (0.00 sec)

-- ROLLBACK 2 INSERTS AND 1 DELETE OPERATIONS
mysql> ROLLBACK;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM Employee;
+--------+---------------+---------+------------+---------+------------+
| EMPNO | ENAME         | JOB     | MANAGER_NO | SAL     | COMMISSION |
+--------+---------------+---------+------------+---------+------------+
|     1 | Kavana Shetty | Manager |       NULL | 5000.00 |    1000.00 |
+--------+---------------+---------+------------+---------+------------+
1 row in set (0.00 sec)
```

You can now see how the rollback operation can be used above.

Adding Constraints

Add Primary Key Constraint

```
-- Add Primary Key Constraint
mysql> ALTER TABLE Employee
    -> ADD CONSTRAINT pk_employee PRIMARY KEY (EMPNO);
Query OK, 0 rows affected (1.65 sec)

-- verify primary key constraint
mysql> DESC Employee;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| EMPNO      | int           | NO   | PRI | NULL    |       |
| ENAME      | varchar(255)  | YES  |     | NULL    |       |
| JOB        | varchar(255)  | YES  |     | NULL    |       |
| MANAGER_NO | int           | YES  |     | NULL    |       |
| SAL        | decimal(10,2) | YES  |     | NULL    |       |
| COMMISSION | decimal(10,2) | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
    -> VALUES (1, 'Ranjan', 'Manager', NULL, 5000.00, 1000.00);
ERROR 1062 (23000): Duplicate entry '1' for key 'Employee.PRIMARY'
```

Since EMPNO field is the primary key it cannot have duplicate values, hence we see that the insert operation fails when provided with a duplicate value.

Add Not Null Constraint

```
-- Add Not Null Constraints
mysql> ALTER TABLE Employee
    -> MODIFY ENAME VARCHAR(255) NOT NULL,
    -> MODIFY JOB VARCHAR(255) NOT NULL,
    -> MODIFY SAL DECIMAL(10, 2) NOT NULL;
```

```
Query OK, 0 rows affected (1.08 sec)

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
    -> VALUES (4, 'Ranjan', 'Manager', NULL, 5000.00, 1000.00);
Query OK, 1 row affected (0.16 sec)

mysql>
mysql> SELECT * FROM Employee;
+-------+---------------+---------+------------+---------+------------+
| EMPNO | ENAME         | JOB     | MANAGER_NO | SAL     | COMMISSION |
+-------+---------------+---------+------------+---------+------------+
|     1 | Kavana Shetty | Manager |       NULL | 5000.00 |    1000.00 |
|     4 | Ranjan        | Manager |       NULL | 5000.00 |    1000.00 |
+-------+---------------+---------+------------+---------+------------+
2 rows in set (0.00 sec)

mysql> INSERT INTO Employee (ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
    -> VALUES (NULL, 'Tester', NULL, 3500.00, NULL);
ERROR 1048 (23000): Column 'ENAME' cannot be null
```

We just illustrated as to how to add not null constraint to the Employee table. We see that the first insert doesn't violate null constraint, however the second insert does violate null constraint as ENAME field cannot be null.

Program 2

Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following.

1. Add a column commission with domain to the Employee table.
2. Insert any five records into the table.
3. Update the column details of job
4. Rename the column of Employ table using alter command.
5. Delete the employee whose Empno is 105.

Solution:

Creating the Employee Table

```
mysql> CREATE DATABASE COMPANY02;
Query OK, 1 row affected (0.16 sec)

mysql> USE COMPANY02;
Database changed

mysql> CREATE TABLE Employee (
    ->     EMPNO INT,
    ->     ENAME VARCHAR(255),
    ->     JOB VARCHAR(255),
    ->     MGR INT,
    ->     SAL DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.48 sec)

mysql> SHOW TABLES;
+--------------------+
| Tables_in_COMPANY02 |
+--------------------+
| Employee           |
+--------------------+
1 row in set (0.00 sec)

mysql> DESC Employee;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| EMPNO | int          | YES  |     | NULL    |       |
| ENAME | varchar(255) | YES  |     | NULL    |       |
| JOB   | varchar(255) | YES  |     | NULL    |       |
| MGR   | int          | YES  |     | NULL    |       |
| SAL   | decimal(10,2) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Adding a Column (Commission) to the Employee Table

```
mysql> ALTER TABLE Employee
    -> ADD COLUMN COMMISSION DECIMAL(10, 2);
Query OK, 0 rows affected (0.37 sec)

mysql> DESC Employee;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| EMPNO      | int          | YES  |     | NULL    |       |
| ENAME      | varchar(255) | YES  |     | NULL    |       |
| JOB        | varchar(255) | YES  |     | NULL    |       |
| MGR        | int          | YES  |     | NULL    |       |
| SAL        | decimal(10,2) | YES  |     | NULL    |       |
| COMMISSION | decimal(10,2) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

We have added a column **COMMISSION** using the **ALTER** command, which is shown above.

Inserting 5 Records into the Employee Table

```
mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, COMMISSION)
    -> VALUES
    ->     (101, 'Radha Bai', 'Manager', NULL, 5000.00, 1000.00),
    ->     (102, 'Krishna Kumar', 'Developer', 101, 4000.00, NULL),
    ->     (103, 'Abdul Sattar', 'Salesperson', 102, 3000.00, 500.00),
    ->     (104, 'Bob Johnson', 'Accountant', 101, 4500.00, NULL),
    ->     (105, 'Amartya Sen', 'HR Manager', 101, 4800.00, 800.00);
Query OK, 5 rows affected (0.12 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Employee;
+-------+--------------+------------+------+---------+------------+
| EMPNO | ENAME        | JOB        | MGR  | SAL     | COMMISSION |
+-------+--------------+------------+------+---------+------------+
|   101 | Radha Bai    | Manager    | NULL | 5000.00 |    1000.00 |
|   102 | Krishna Kumar | Developer  |  101 | 4000.00 |       NULL |
|   103 | Abdul Sattar | Salesperson |  102 | 3000.00 |     500.00 |
|   104 | Bob Johnson  | Accountant |  101 | 4500.00 |       NULL |
|   105 | Amartya Sen  | HR Manager |  101 | 4800.00 |     800.00 |
+-------+--------------+------------+------+---------+------------+
5 rows in set (0.00 sec)
```

Updating Column Details (JOB) in the Employee Table

```
mysql> UPDATE Employee
    -> SET JOB = 'Senior Developer'
    -> WHERE EMPNO = 102;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Employee;
+-------+--------------+-----------------+------+---------+------------+
| EMPNO | ENAME        | JOB             | MGR  | SAL     | COMMISSION |
+-------+--------------+-----------------+------+---------+------------+
|   101 | Radha Bai    | Manager         | NULL | 5000.00 |    1000.00 |
|   102 | Krishna Kumar | Senior Developer |  101 | 4000.00 |       NULL |
|   103 | Abdul Sattar | Salesperson     |  102 | 3000.00 |     500.00 |
|   104 | Bob Johnson  | Accountant      |  101 | 4500.00 |       NULL |
|   105 | Amartya Sen  | HR Manager      |  101 | 4800.00 |     800.00 |
+-------+--------------+-----------------+------+---------+------------+
5 rows in set (0.00 sec)
```

Renaming a Column in the Employee Table

To rename the `MGR` column to `MANAGER_ID`:

```
mysql> ALTER TABLE Employee
    -> CHANGE COLUMN MGR MANAGER_ID INT;
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> DESC Employee;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| EMPNO      | int           | YES  |     | NULL    |       |
| ENAME      | varchar(255)  | YES  |     | NULL    |       |
| JOB        | varchar(255)  | YES  |     | NULL    |       |
| MANAGER_ID | int           | YES  |     | NULL    |       |
| SAL        | decimal(10,2) | YES  |     | NULL    |       |
| COMMISSION | decimal(10,2) | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Deleting a Specific Employee (EMPNO = 105) from the Employee Table

```
mysql> DELETE FROM Employee
    -> WHERE EMPNO = 105;
Query OK, 1 row affected (0.14 sec)

mysql> SELECT * FROM Employee;
+-------+--------------+-----------------+------------+---------+------------+
| EMPNO | ENAME        | JOB             | MANAGER_ID | SAL     | COMMISSION |
+-------+--------------+-----------------+------------+---------+------------+
|   101 | Radha Bai    | Manager         |       NULL | 5000.00 |    1000.00 |
|   102 | Krishna Kumar | Senior Developer |        101 | 4000.00 |       NULL |
|   103 | Abdul Sattar | Salesperson     |        102 | 3000.00 |     500.00 |
|   104 | Bob Johnson  | Accountant      |        101 | 4500.00 |       NULL |
+-------+--------------+-----------------+------------+---------+------------+
4 rows in set (0.00 sec)
```

Program 3

Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by,Orderby.

Employee(E_id, E_name, Age, Salary)

1. Create Employee table containing all Records E_id, E_name, Age, Salary.
2. Count number of employee names from employeetable
3. Find the Maximum age from employee table.
4. Find the Minimum age from employeetable.
5. Find salaries of employee in Ascending Order.
6. Find grouped salaries of employees

Solution:

1. Creating the Employee Table

```
mysql> CREATE DATABASE COMPANY03;
Query OK, 1 row affected (0.09 sec)

mysql> USE COMPANY03;
Database changed

mysql> CREATE TABLE Employee (
    ->     E_id INT PRIMARY KEY,
    ->     E_name VARCHAR(255),
    ->     Age INT,
    ->     Salary DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (1.00 sec)

mysql> DESC Employee;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| E_id   | int           | NO   | PRI | NULL    |       |
| E_name | varchar(255)  | YES  |     | NULL    |       |
| Age    | int           | YES  |     | NULL    |       |
| Salary | decimal(10,2) | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

2. Populating the Employee Table with 12 Records

```
mysql> INSERT INTO Employee (E_id, E_name, Age, Salary)
    -> VALUES
    ->     (1, 'Samarth', 30, 50000.00),
    ->     (2, 'Ramesh Kumar', 25, 45000.00),
    ->     (3, 'Seema Banu', 35, 60000.00),
    ->     (4, 'Dennis Anil', 28, 52000.00),
    ->     (5, 'Rehman Khan', 32, 58000.00),
    ->     (6, 'Pavan Gowda', 40, 70000.00),
    ->     (7, 'Shruthi Bhat', 27, 48000.00),
    ->     (8, 'Sandesh Yadav', 29, 51000.00),
    ->     (9, 'Vikram Acharya', 33, 62000.00),
    ->     (10, 'Praveen Bellad', 26, 46000.00),
    ->     (11, 'Sophia Mary', 31, 55000.00),
    ->     (12, 'Darshan Desai', 34, 63000.00);
Query OK, 12 rows affected (0.14 sec)
Records: 12  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Employee;
+------+---------------+------+----------+
| E_id | E_name        | Age  | Salary   |
+------+---------------+------+----------+
|    1 | Samarth       |   30 | 50000.00 |
|    2 | Ramesh Kumar  |   25 | 45000.00 |
|    3 | Seema Banu    |   35 | 60000.00 |
|    4 | Dennis Anil   |   28 | 52000.00 |
|    5 | Rehman Khan   |   32 | 58000.00 |
|    6 | Pavan Gowda   |   40 | 70000.00 |
```

```
|   7 | Shruthi Bhat   |  27 | 48000.00 |
|   8 | Sandesh Yadav  |  29 | 51000.00 |
|   9 | Vikram Acharya |  33 | 62000.00 |
|  10 | Praveen Bellad |  26 | 46000.00 |
|  11 | Sophia Mary    |  31 | 55000.00 |
|  12 | Darshan Desai  |  34 | 63000.00 |
+-----+----------------+-----+----------+
12 rows in set (0.00 sec)
```

3. Count Number of Employee Names

```
mysql> SELECT COUNT(E_name) AS TotalEmployees
    -> FROM Employee;
+----------------+
| TotalEmployees |
+----------------+
|             12 |
+----------------+
1 row in set (0.00 sec)
```

4. Find the Maximum Age

```
mysql> SELECT MAX(Age) AS MaxAge
    -> FROM Employee;
+--------+
| MaxAge |
+--------+
|     40 |
+--------+
1 row in set (0.01 sec)
```

5. Find the Minimum Age

```
mysql> SELECT MIN(Age) AS MinAge
    -> FROM Employee;
+--------+
| MinAge |
+--------+
|     25 |
+--------+
1 row in set (0.00 sec)
```

6. Find Salaries of Employees in Ascending Order

```
mysql> SELECT E_name, Salary
    -> FROM Employee
    -> ORDER BY Salary ASC;
+----------------+----------+
| E_name         | Salary   |
+----------------+----------+
| Ramesh Kumar   | 45000.00 |
```

```
| Praveen Bellad | 46000.00 |
| Shruthi Bhat   | 48000.00 |
| Samarth        | 50000.00 |
| Dennis Anil    | 52000.00 |
| Sandesh Yadav  | 52000.00 |
| Sophia Mary    | 55000.00 |
| Rehman Khan    | 58000.00 |
| Seema Banu     | 62000.00 |
| Vikram Acharya | 62000.00 |
| Darshan Desai  | 63000.00 |
| Pavan Gowda    | 70000.00 |
+----------------+----------+
12 rows in set (0.00 sec)
```

7. Find Grouped Salaries of Employees

```
mysql> SELECT Salary, COUNT(*) AS EmployeeCount
    -> FROM Employee
    -> GROUP BY Salary;
+----------+---------------+
| Salary   | EmployeeCount |
+----------+---------------+
| 50000.00 |             1 |
| 45000.00 |             1 |
| 62000.00 |             2 |
| 52000.00 |             2 |
| 58000.00 |             1 |
| 70000.00 |             1 |
| 48000.00 |             1 |
| 46000.00 |             1 |
| 55000.00 |             1 |
| 63000.00 |             1 |
+----------+---------------+
10 rows in set (0.00 sec)
```

In these queries:

- `COUNT(E_name)` counts the number of non-NULL values in the `E_name` column.
- `MAX(Age)` finds the maximum age among the employees.
- `MIN(Age)` finds the minimum age among the employees.
- `ORDER BY Salary ASC` sorts the employees based on their salaries in ascending order.
- `GROUP BY Salary` groups employees by their salaries and counts the number of employees for each salary.

Program 4

Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.
CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)
Solution:

1. Create the `CUSTOMERS` Table

First, create the **`CUSTOMERS`** table with the specified columns:

```
mysql> CREATE DATABASE COMPANY04;
Query OK, 1 row affected (0.14 sec)

mysql> USE COMPANY04;
```

```
Database changed

mysql> CREATE TABLE CUSTOMERS (
    ->     ID INT PRIMARY KEY AUTO_INCREMENT,
    ->     NAME VARCHAR(255),
    ->     AGE INT,
    ->     ADDRESS VARCHAR(255),
    ->     SALARY DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.49 sec)
```

To achieve the desired functionality of capturing changes on `INSERT`, `UPDATE`, or `DELETE` operations and displaying the salary difference in MySQL, you'll need to create separate row-level triggers for each operation (`INSERT`, `UPDATE`, `DELETE`). These triggers will capture the `OLD` and `NEW` values of the `SALARY` column and display the salary difference when an INSERT, UPDATE, or DELETE operation occurs.Here's how you can do it:

2. Create Trigger for INSERT Operation

```
-- INSERT TRIGGER
DELIMITER //

CREATE TRIGGER after_insert_salary_difference
AFTER INSERT ON CUSTOMERS
FOR EACH ROW
BEGIN
            SET @my_sal_diff = CONCAT('salary inserted is ', NEW.SALARY);
END;//

DELIMITER ;
```

3. Create Trigger for UPDATE Operation

```
-- UPDATE TRIGGER
DELIMITER //

CREATE TRIGGER after_update_salary_difference
AFTER UPDATE ON CUSTOMERS
FOR EACH ROW
BEGIN
    DECLARE old_salary DECIMAL(10, 2);
    DECLARE new_salary DECIMAL(10, 2);

    SET old_salary = OLD.SALARY;
    SET new_salary = NEW.SALARY;
            SET @my_sal_diff = CONCAT('salary difference after update is ', NEW.SALARY -
OLD.SALARY);
END;//

DELIMITER ;
```

4. Create Trigger for DELETE Operation

```
-- DELETE TRIGGER
DELIMITER //

CREATE TRIGGER after_delete_salary_difference
AFTER DELETE ON CUSTOMERS
FOR EACH ROW
BEGIN
        SET @my_sal_diff = CONCAT('salary deleted is ', OLD.SALARY);
END;//

DELIMITER ;
```

5. Testing the Trigger:

Once the triggers are created, you can perform `INSERT`, `UPDATE`, or `DELETE` operations on the `CUSTOMERS` table to observe the salary difference messages generated by the triggers.

For example:

```
mysql> -- test INSERT TRIGGER
mysql> INSERT INTO CUSTOMERS (NAME, AGE, ADDRESS, SALARY)
    -> VALUES ('Shankara', 35, '123 Main St', 50000.00);
Query OK, 1 row affected (0.14 sec)

mysql>
mysql> SELECT @my_sal_diff AS SAL_DIFF;
+---------------------------+
| SAL_DIFF                  |
+---------------------------+
| salary inserted is 50000.00 |
+---------------------------+
1 row in set (0.00 sec)

mysql> -- test UPDATE TRIGGER
mysql> UPDATE CUSTOMERS
    -> SET SALARY = 55000.00
    -> WHERE ID = 1;
Query OK, 1 row affected (0.13 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT @my_sal_diff AS SAL_DIFF;
+----------------------------------------+
| SAL_DIFF                               |
+----------------------------------------+
| salary difference after update is 5000.00 |
+----------------------------------------+
1 row in set (0.00 sec)

mysql> -- test DELETE TRIGGER
mysql> DELETE FROM CUSTOMERS
    -> WHERE ID = 1;
Query OK, 1 row affected (0.13 sec)

mysql>
```

```
mysql> SELECT @my_sal_diff AS SAL_DIFF;
+--------------------------+
| SAL_DIFF                 |
+--------------------------+
| salary deleted is 55000.00 |
+--------------------------+
1 row in set (0.00 sec)
```

Each operation (`INSERT`, `UPDATE`, `DELETE`) will trigger the respective trigger (`after_insert_salary_difference`, `after_update_salary_difference`, `after_delete_salary_difference`), which will display the salary change or difference associated with that operation.

By using separate triggers for each operation and utilizing the `OLD` and `NEW` keywords appropriately within the trigger bodies, you can effectively capture and handle changes to the `SALARY` column in the `CUSTOMERS` table in MySQL. You can adjust the trigger logic and message formatting as needed based on your specific requirements.

Program 5

Create cursor for Employee table & extract the values from the table. Declare the variables ,Open the cursor & extrct the values from the cursor. Close the cursor.

Employee(E_id, E_name, Age, Salary)

Solution:

1. Creating the Employee Table and insert few records

```
CREATE DATABASE COMPANY05;

USE COMPANY05;

CREATE TABLE Employee (
    E_id INT,
    E_name VARCHAR(255),
    Age INT,
    Salary DECIMAL(10, 2)
);
```

```
INSERT INTO Employee (E_id, E_name, Age, Salary)
VALUES
    (1, 'Samarth', 30, 50000.00),
    (2, 'Ramesh Kumar', 25, 45000.00),
    (3, 'Seema Banu', 35, 62000.00),
    (4, 'Dennis Anil', 28, 52000.00),
    (5, 'Rehman Khan', 32, 58000.00);
```

2. Create a Stored Procedure with Cursor

To create a cursor for the `Employee` table, extract values using the cursor, and then close the cursor in MySQL, you'll need to use stored procedures that support cursor operations.

```
DELIMITER //

CREATE PROCEDURE fetch_employee_data()
BEGIN
    -- Declare variables to store cursor values
    DECLARE emp_id INT;
    DECLARE emp_name VARCHAR(255);
    DECLARE emp_age INT;
    DECLARE emp_salary DECIMAL(10, 2);

    -- Declare a cursor for the Employee table
    DECLARE emp_cursor CURSOR FOR
        SELECT E_id, E_name, Age, Salary
        FROM Employee;

    -- Declare a continue handler for the cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET @finished = 1;

    -- Open the cursor
    OPEN emp_cursor;

    -- Initialize a variable to control cursor loop
    SET @finished = 0;

    -- Loop through the cursor results
    cursor_loop: LOOP
        -- Fetch the next row from the cursor into variables
        FETCH emp_cursor INTO emp_id, emp_name, emp_age, emp_salary;

        -- Check if no more rows to fetch
        IF @finished = 1 THEN
            LEAVE cursor_loop;
        END IF;

        -- Output or process each row (for demonstration, print the values)
        SELECT CONCAT('Employee ID: ', emp_id, ', Name: ', emp_name, ', Age: ', emp_age, ', Salary: ',
emp_salary) AS Employee_Info;
    END LOOP;

    -- Close the cursor
```

```
    CLOSE emp_cursor;
END//

DELIMITER ;
```

In this stored procedure (`fetch_employee_data`):

- We declare variables (`emp_id`, `emp_name`, `emp_age`, `emp_salary`) to store values retrieved from the cursor.
- A cursor (`emp_cursor`) is declared to select `E_id`, `E_name`, `Age`, and `Salary` from the `Employee` table.
- We declare a continue handler (`CONTINUE HANDLER`) for `NOT FOUND` condition to handle the end of cursor data.
- The cursor is opened (`OPEN emp_cursor`), and a loop (`cursor_loop`) is used to fetch each row from the cursor.
- We fetch values into the variables and process them within the loop (for demonstration, we print the values using a `SELECT` statement).
- The loop continues until all rows are fetched (`@finished = 1`).
- Finally, the cursor is closed (`CLOSE emp_cursor`).

3. Execute the Stored Procedure

Once the stored procedure `fetch_employee_data` is created, you can execute it to fetch and process data from the `Employee` table:

```
mysql> CALL fetch_employee_data();
+-----------------------------------------------------+
| Employee_Info                                       |
+-----------------------------------------------------+
| Employee ID: 1, Name: Samarth, Age: 30, Salary: 50000.00 |
+-----------------------------------------------------+
1 row in set (0.07 sec)

+-----------------------------------------------------+
| Employee_Info                                       |
+-----------------------------------------------------+
| Employee ID: 2, Name: Ramesh Kumar, Age: 25, Salary: 45000.00 |
+-----------------------------------------------------+
1 row in set (0.07 sec)

+-----------------------------------------------------+
| Employee_Info                                       |
+-----------------------------------------------------+
| Employee ID: 3, Name: Seema Banu, Age: 35, Salary: 62000.00 |
+-----------------------------------------------------+
1 row in set (0.07 sec)

+-----------------------------------------------------+
| Employee_Info                                       |
+-----------------------------------------------------+
| Employee ID: 4, Name: Dennis Anil, Age: 28, Salary: 52000.00 |
```

```
+-----------------------------------------------------------+
1 row in set (0.07 sec)

+-----------------------------------------------------------+
| Employee_Info                                             |
+-----------------------------------------------------------+
| Employee ID: 5, Name: Rehman Khan, Age: 32, Salary: 58000.00 |
+-----------------------------------------------------------+
1 row in set (0.07 sec)

Query OK, 0 rows affected (0.07 sec)
```

- The stored procedure `fetch_employee_data` declares variables (`emp_id`, `emp_name`, `emp_age`, `emp_salary`) to store values retrieved from the cursor.
- A cursor (`emp_cursor`) is declared for the `Employee` table to select `E_id`, `E_name`, `Age`, and `Salary`.
- The cursor is opened (`OPEN emp_cursor`), and the `FETCH` statement retrieves the first row from the cursor into the declared variables.
- A `WHILE` loop processes each row fetched by the cursor (`SQLSTATE() = '00000'` checks for successful fetching).
- Within the loop, you can perform operations or output the values of each row.
- The `CLOSE` statement closes the cursor after processing all rows.

This example demonstrates how to create and use a cursor in MySQL to extract values from the `Employee` table row by row. Adjust the cursor query and processing logic based on your table structure and desired operations.

Program 6

Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Solution:

To accomplish this task in MySQL, we can use a stored procedure with a parameterized cursor to merge data from one table (`N_RollCall`) into another table (`O_RollCall`) while skipping existing data. We'll iterate through the records of `N_RollCall` and insert them into `O_RollCall` only if they do not already exist.

1. Create the Tables

First, let's create the `N_RollCall` and `O_RollCall` tables with similar structure:

```
CREATE DATABASE ROLLCALL;
```

```
USE ROLLCALL;

-- Create N_RollCall table
CREATE TABLE N_RollCall (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(255),
    birth_date DATE
);

-- Create O_RollCall table with common data
CREATE TABLE O_RollCall (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(255),
    birth_date DATE
);
```

2. Add Sample Records to both tables

Let's insert some sample data into the `O_RollCall` table:

```
mysql> -- Insert common data into O_RollCall
mysql> INSERT INTO O_RollCall (student_id, student_name, birth_date)
    -> VALUES
    ->     (1, 'Shivanna', '1995-08-15'),
    ->     (3, 'Cheluva', '1990-12-10');
Query OK, 2 rows affected (0.17 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Let's insert some sample data into the `N_RollCall` table, including records that are common with `O_RollCall`:

```
mysql> -- Insert sample records into N_RollCall
mysql> INSERT INTO N_RollCall (student_id, student_name, birth_date)
    -> VALUES
    ->     (1, 'Shivanna', '1995-08-15'),  -- Common record with O_RollCall
    ->     (2, 'Bhadramma', '1998-03-22'),
    ->     (3, 'Cheluva', '1990-12-10'), -- Common record with O_RollCall
    ->     (4, 'Devendra', '2000-05-18'),
    ->     (5, 'Eshwar', '1997-09-03');
Query OK, 5 rows affected (0.21 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

3. Define the Stored Procedure

Next, let's define the `merge_rollcall_data` stored procedure to merge records from `N_RollCall` into `O_RollCall`, skipping existing records:

```
DELIMITER //

CREATE PROCEDURE merge_rollcall_data()
BEGIN
    DECLARE done INT DEFAULT FALSE;
```

```
    DECLARE n_id INT;
    DECLARE n_name VARCHAR(255);
    DECLARE n_birth_date DATE;

    -- Declare cursor for N_RollCall table
    DECLARE n_cursor CURSOR FOR
        SELECT student_id, student_name, birth_date
        FROM N_RollCall;

    -- Declare handler for cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET done = TRUE;

    -- Open the cursor
    OPEN n_cursor;

    -- Start looping through cursor results
    cursor_loop: LOOP
        -- Fetch data from cursor into variables
        FETCH n_cursor INTO n_id, n_name, n_birth_date;

        -- Check if no more rows to fetch
        IF done THEN
            LEAVE cursor_loop;
        END IF;

        -- Check if the data already exists in O_RollCall
        IF NOT EXISTS (
            SELECT 1
            FROM O_RollCall
            WHERE student_id = n_id
        ) THEN
            -- Insert the record into O_RollCall
            INSERT INTO O_RollCall (student_id, student_name, birth_date)
            VALUES (n_id, n_name, n_birth_date);
        END IF;
    END LOOP;

    -- Close the cursor
    CLOSE n_cursor;
END//

DELIMITER ;
```

- The stored procedure `merge_rollcall_data` uses a cursor (`n_cursor`) to iterate through the records of the `N_RollCall` table.
- Inside the cursor loop (`cursor_loop`), each record (`n_id`, `n_name`, `n_date`) from `N_RollCall` is fetched and checked against the `O_RollCall` table.
- If the record does not already exist in `O_RollCall` (checked using `NOT EXISTS`), it is inserted into `O_RollCall`.
- The cursor loop continues until all records from `N_RollCall` have been processed.
- The cursor is then closed (`CLOSE n_cursor`).

4. Execute the Stored Procedure

Finally, execute the `merge_rollcall_data` stored procedure to merge records from `N_RollCall` into `O_RollCall` while skipping existing records:

```
mysql> CALL merge_rollcall_data();
Query OK, 0 rows affected (0.87 sec)
```

5. Verify Records in `O_RollCall`

After executing the procedure, verify the records in the `O_RollCall` table to confirm that new records from `N_RollCall` have been inserted, while existing common records have been skipped:

```
mysql> -- Select all records from O_RollCall
mysql> SELECT * FROM O_RollCall;
+------------+--------------+------------+
| student_id | student_name | birth_date |
+------------+--------------+------------+
|          1 | Shivanna     | 1995-08-15 |<-- Common record, not duplicated
|          2 | Bhadramma    | 1998-03-22 |<-- New record from N_RollCall
|          3 | Cheluva      | 1990-12-10 |<-- Common record, not duplicated
|          4 | Devendra     | 2000-05-18 |<-- New record from N_RollCall
|          5 | Eshwar       | 1997-09-03 |<-- New record from N_RollCall
+------------+--------------+------------+
5 rows in set (0.00 sec)
```

Program 7

Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.

Solution:

1. Installing Open Source NoSQL Data base MongoDB

Please refer to the blog below which contains detailed procedure of installing Open Source NoSQL Data base MongoDB.

2. Perform basic CRUD(Create, Read, Update & Delete) operations.

1. Start MongoDB.

Launch the MongoDB daemon using the following command:

```
sudo systemctl start mongod
```

2. Start the MongoDB Shell

Launch the MongoDB shell to perform basic CRUD operations.

```
mongosh
```

3. Switch to a Database (Optional):

If you want to use a specific database, switch to that database using the `use` command. If the database doesn't exist, MongoDB will create it implicitly when you insert data into it:

```
test> use bookDB
switched to db bookDB
bookDB>
```

4. Create the `ProgrammingBooks` Collection:

To create the `ProgrammingBooks` collection, use the `createCollection()` method. This step is optional because MongoDB will automatically create the collection when you insert data into it, but you can explicitly create it if needed:

```
bookDB> db.createCollection("ProgrammingBooks")
```

This command will create an empty `ProgrammingBooks` collection in the current database (`bookDB`).

5. INSERT operations

a. Insert 5 Documents into the `ProgrammingBooks` Collection :

Now, insert 5 documents representing programming books into the `ProgrammingBooks` collection using the `insertMany()` method:

```
bookDB> db.ProgrammingBooks.insertMany([
  {
    title: "Clean Code: A Handbook of Agile Software Craftsmanship",
    author: "Robert C. Martin",
    category: "Software Development",
    year: 2008
  },
  {
    title: "JavaScript: The Good Parts",
    author: "Douglas Crockford",
    category: "JavaScript",
    year: 2008
  },
```

```
  {
    title: "Design Patterns: Elements of Reusable Object-Oriented Software",
    author: "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides",
    category: "Software Design",
    year: 1994
  },
  {
    title: "Introduction to Algorithms",
    author: "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein",
    category: "Algorithms",
    year: 1990
  },
  {
    title: "Python Crash Course: A Hands-On, Project-Based Introduction to Programming",
    author: "Eric Matthes",
    category: "Python",
    year: 2015
  }
])
```

b. Insert a Single Document into `ProgrammingBooks`:

Use the **`insertOne()`** method to insert a new document into the **`ProgrammingBooks`** collection:

```
bookDB> db.ProgrammingBooks.insertOne({
  title: "The Pragmatic Programmer: Your Journey to Mastery",
  author: "David Thomas, Andrew Hunt",
  category: "Software Development",
  year: 1999
})
```

6. Read (Query) Operations

a. Find All Documents

To retrieve all documents from the **`ProgrammingBooks`** collection:

```
bookDB> db.ProgrammingBooks.find().pretty()
[
  {
    _id: ObjectId('663eaaebae582498972202df'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e0'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
```

```
  _id: ObjectId('663eaaebae582498972202e1'),
  title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
  author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
  category: 'Software Design',
  year: 1994
 },
 {
  _id: ObjectId('663eaaebae582498972202e2'),
  title: 'Introduction to Algorithms',
  author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein',
  category: 'Algorithms',
  year: 1990
 },
 {
  _id: ObjectId('663eaaebae582498972202e3'),
  title: 'Python Crash Course: A Hands-On, Project-Based Introduction to Programming',
  author: 'Eric Matthes',
  category: 'Python',
  year: 2015
 },
 {
  _id: ObjectId('663eab05ae582498972202e4'),
  title: 'The Pragmatic Programmer: Your Journey to Mastery',
  author: 'David Thomas, Andrew Hunt',
  category: 'Software Development',
  year: 1999
 }
]
```

b. Find Documents Matching a Condition

To find books published after the year 2000:

```
bookDB> db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()
[
 {
  _id: ObjectId('663eaaebae582498972202df'),
  title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
  author: 'Robert C. Martin',
  category: 'Software Development',
  year: 2008
 },
 {
  _id: ObjectId('663eaaebae582498972202e0'),
  title: 'JavaScript: The Good Parts',
  author: 'Douglas Crockford',
  category: 'JavaScript',
  year: 2008
 },
 {
  _id: ObjectId('663eaaebae582498972202e3'),
  title: 'Python Crash Course: A Hands-On, Project-Based Introduction to Programming',
  author: 'Eric Matthes',
  category: 'Python',
  year: 2015
 }
]
```

7. Update Operations

a. Update a Single Document

To update a specific book (e.g., change the author of a book):

```
bookDB>db.ProgrammingBooks.updateOne(
  { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
  { $set: { author: "Robert C. Martin (Uncle Bob)" } }
)

//verify by displaying books published in year 2008
bookDB> db.ProgrammingBooks.find({ year: { $eq: 2008 } }).pretty()
[
  {
    _id: ObjectId('663eaaebae582498972202df'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e0'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  }
]
```

b. Update Multiple Documents

To update multiple books (e.g., update the category of books published before 2010):

```
bookDB> db.ProgrammingBooks.updateMany(
  { year: { $lt: 2010 } },
  { $set: { category: "Classic Programming Books" } }
)

//verify the update operation by displaying books published before year 2010
bookDB> db.ProgrammingBooks.find({ year: { $lt: 2010 } }).pretty()
[
  {
    _id: ObjectId('663eaaebae582498972202df'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e0'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'Classic Programming Books',
```

```
    year: 2008
  },
  {
    _id: ObjectId('663eaaebae582498972202e1'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Classic Programming Books',
    year: 1994
  },
  {
    _id: ObjectId('663eaaebae582498972202e2'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein',
    category: 'Classic Programming Books',
    year: 1990
  },
  {
    _id: ObjectId('663eab05ae582498972202e4'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Classic Programming Books',
    year: 1999
  }
]
```

8. Delete Operations

a. Delete a Single Document

To delete a specific book from the collection (e.g., delete a book by title):

```
bookDB> db.ProgrammingBooks.deleteOne({ title: "JavaScript: The Good Parts" })
{ acknowledged: true, deletedCount: 1 }
```

You can check whether the specified document is deleted by displaying the contents of the collection.

b. Delete Multiple Documents

To delete multiple books based on a condition (e.g., delete all books published before 1995):

```
bookDB> db.ProgrammingBooks.deleteMany({ year: { $lt: 1995 } })
{ acknowledged: true, deletedCount: 2 }
```

You can check whether the specified documents were deleted by displaying the contents of the collection.

c. Delete All Documents in the Collection:

To delete all documents in a collection (e.g., `ProgrammingBooks`), use the `deleteMany()` method with an empty filter `{}`:

```
//delete all documents in a collection
bookDB> db.ProgrammingBooks.deleteMany({})
{ acknowledged: true, deletedCount: 3 }

//verify by displaying the collection
bookDB> db.ProgrammingBooks.find().pretty()
```

9. Delete the Collection Using drop():

To delete a collection named `ProgrammingBooks`, use the `drop()` method with the name of the collection:

```
bookDB> show collections
ProgrammingBooks

bookDB> db.ProgrammingBooks.drop()
true

bookDB> show collections

bookDB>
```

The command `db.ProgrammingBooks.drop( )` will permanently delete the `ProgrammingBooks` collection from the current database (`bookDB`).

After deleting the collection, you can verify that it no longer exists by listing all collections in the database using the command `show collections`.