Database Management System Lab Component

Question 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 the user.
2. Insert 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

1. *--creating a DataBase*

```
sudo mysql -u root
CREATE DATABASE COMPANY;
USE COMPANY;

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

--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';
```

2.
```
mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
   -> VALUES (1, 'Kavana Shetty', 'Manager', NULL, 5000.00, 1000.00);
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);
```

3.
```
mysql> ALTER TABLE Employee
   -> ADD CONSTRAINT pk_employee PRIMARY KEY (EMPNO);
```

```
-- 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;

5.   mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
    -> VALUES (4, 'Ranjan', 'Manager', NULL, 5000.00, 1000.00);
```

Question 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.

Solution

1.
```
mysql> CREATE DATABASE COMPANY02;

    mysql> USE COMPANY02;

    mysql> CREATE TABLE Employee (
  ->    EMPNO INT,
  ->    ENAME VARCHAR(255),
  ->    JOB VARCHAR(255),
  ->    MGR INT,
  ->    SAL DECIMAL(10, 2)
  -> );
--Add a column
mysql> ALTER TABLE Employee
  -> ADD COLUMN COMMISSION DECIMAL(10, 2);

mysql> DESC Employee;
```

2.
```
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);
```

3. mysql> UPDATE Employee
-> SET JOB = 'Senior Developer'
-> WHERE EMPNO = 102;

4. mysql> ALTER TABLE Employee
-> CHANGE COLUMN MGR MANAGER_ID INT;

5. mysql> DELETE FROM Employee
-> WHERE EMPNO = 105;

Question 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 Employee table
3. Find the Maximum age from Employee table.
4. Find the Minimum age from Employee table.
5. Find salaries of employee in Ascending Order.
6. Find grouped salaries of employees.

1. mysql> CREATE DATABASE COMPANY03;

```
mysql> USE COMPANY03;

mysql> CREATE TABLE Employee (
   ->    E_id INT PRIMARY KEY,
   ->    E_name VARCHAR(255),
   ->    Age INT,
   ->    Salary DECIMAL(10, 2)
   -> );

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);
```

2.
```
mysql> SELECT COUNT(E_name) AS TotalEmployees
-> FROM Employee;
```

3.
```
mysql> SELECT MAX(Age) AS MaxAge
-> FROM Employee;
```

4.
```
mysql> SELECT MIN(Age) AS MinAge
-> FROM Employee;
```

5.
```
mysql> SELECT E_name, Salary
-> FROM Employee
-> ORDER BY Salary ASC;
```

6.
```
mysql> SELECT Salary, COUNT(*) AS EmployeeCount
-> FROM Employee
-> GROUP BY Salary;
```

Question 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)

```
mysql> CREATE DATABASE COMPANY04;

mysql> USE COMPANY04;

mysql> CREATE TABLE CUSTOMERS (
   ->    ID INT PRIMARY KEY AUTO_INCREMENT,
   ->    NAME VARCHAR(255),
   ->    AGE INT,
   ->    ADDRESS VARCHAR(255),
   ->    SALARY DECIMAL(10, 2)
   -> );
```

```
-- 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 ;


    -- 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 ;

    -- 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 ;
```

```
mysql> -- test INSERT TRIGGER
mysql> INSERT INTO CUSTOMERS (NAME, AGE, ADDRESS, SALARY)
   -> VALUES ('Shankara', 35, '123 Main St', 50000.00);

mysql> SELECT @my_sal_diff AS SAL_DIFF;
+----------------------------+
| SAL_DIFF                   |
+----------------------------+
| salary inserted is 50000.00 |
+----------------------------+

mysql> -- test UPDATE TRIGGER
mysql> UPDATE CUSTOMERS
   -> SET SALARY = 55000.00
   -> WHERE ID = 1;

mysql> SELECT @my_sal_diff AS SAL_DIFF;
+-------------------------------------------+
| SAL_DIFF                                  |
+-------------------------------------------+
| salary difference after update is 5000.00 |
+-------------------------------------------+

mysql> -- test DELETE TRIGGER
mysql> DELETE FROM CUSTOMERS
   -> WHERE ID = 1;

mysql> SELECT @my_sal_diff AS SAL_DIFF;
+----------------------------+
| SAL_DIFF                   |
+----------------------------+
| salary deleted is 55000.00 |
+----------------------------+
```

Question 5:

Create cursor for Employee table & extract the values from the table. Declare the variables,Open the cursor & extract the values from the cursor. Close the cursor.
CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)

```
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);

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 ;

mysql> CALL fetch_employee_data();
```

Question 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.

```
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
);

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');

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');

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 ;

mysql> CALL merge_rollcall_data();

mysql> -- Select all records from O_RollCall
mysql> SELECT * FROM O_RollCall;
```

Question 7:

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

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

https://moodle.sit.ac.in/blog

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**.