JYOTHY INSTITUTE OF TECHNOLOGY

Tataguni, Off Kanakapura Road, Bengaluru – 560 082

APPROVED BY AICTE & AFFILIATED TO VTU, BELAGAVI

Department of

Information Science & Engineering

Accredited by NBA, New Delhi

Lab Manual

DATABASE MANAGEMENT SYSTEM LABORATORY

(BCS403)

Signature of the Faculty

Signature of the HoD

Institution Vision & Mission

Vision:

"To be an Institution of Excellence in Engineering education, Innovation and Research and work towards evolving great leaders for the country's future and meeting global needs."

Mission:

The Institution aims at providing a vibrant, intellectually and emotionally rich teaching learning environment with the State of the Art Infrastructure and recognizing and nurturing the potential of each individual to evolve into ones own self and contribute to the welfare of all.

Department Vision & Mission

Vision:

"To be a center of excellence in Computer Science and Engineering education, focus on research, innovation and entrepreneurial skill development with professional competency."

Mission:

M1: To provide state of the art ICT infrastructure and innovative, research-oriented teaching-learning environment and motivation for self-learning & problem-solving abilities by recruiting committed faculty.

M2: To encourage Industry Institute Interaction & multi-disciplinary approach for problem-solving and adapt to ever-changing global IT trends.

M3: To imbibe awareness of societal responsibility and leadership qualities with professional competency and ethics.

DATABASE MANAGEMENT SYSTEM		Semester	4
Course Code	**BCS403**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 hours Theory + 8-10 Lab slots	Total Marks	100
Credits	04	Exam Hours	
Examination nature (SEE)	Theory		

Course objectives:

- To Provide a strong foundation in database concepts, technology, and practice.
- To Practice SQL programming through a variety of database problems.
- To Understand the relational database design principles.
- To Demonstrate the use of concurrency and transactions in database.
- To Design and build database applications for real world problems.
- To become familiar with database storage structures and access techniques.

Teaching-Learning Process

These are sample Strategies, which teachers can use to accelerate the attainment of the variouscourse outcomes.

1. Lecturer method (L) needs not to be only a traditional lecture method, but alternative effectiveteaching methods could be adopted to attain the outcomes.
2. Use of Video/Animation to explain functioning of various concepts.
3. Encourage collaborative (Group Learning) Learning in the class.
4. Ask at least three HOT (Higher order Thinking) questions in the class, which promotes criticalthinking.
5. Adopt Problem Based Learning (PBL), which fosters students' Analytical skills, develop design thinking skills such as the ability to design, evaluate, generalize, and analyze information rather thansimply recall it.
6. Introduce Topics in manifold representations.
7. Show the different ways to solve the same problem with different circuits/logic and encourage thestudents to come up with their own creative ways to solve them.
8. Discuss how every concept can be applied to the real world - and when that's possible, it helpsimprove the students' understanding
9. Use any of these methods: Chalk and board, Active Learning, Case Studies

MODULE-1	**No. of Hours: 8**

Introduction to Databases: Introduction, Characteristics of database approach, Advantages of using theDBMS approach, History of database applications.

Overview of Database Languages and Architectures: Data Models, Schemas, and Instances. Three schema architecture and data independence, database languages, and interfaces, The Database System environment.**Conceptual Data Modelling using Entities and Relationships:** Entity types, Entity sets and structuralconstraints, Weak entity types, ER diagrams, Specialization and Generalization.

Textbook 1:Ch 1.1 to 1.8, 2.1 to 2.6, 3.1 to 3.10RBT: L1, L2, L3

MODULE-2	**No. of Hours: 8**

Relational Model: Relational Model Concepts, Relational Model Constraints and relational database schemas,Update operations, transactions, and dealing with constraint violations.
Relational Algebra: Unary and Binary relational operations, additional relational operations (aggregate,grouping, etc.) Examples of Queries in relational algebra.
Mapping Conceptual Design into a Logical Design: Relational Database Design using ER-to-Relationalmapping.

Textbook 1: Ch 5.1 to 5.3, Ch 8.1 to 8.5; Ch 9.1 to 9.2 Textbook 2: 3.5RBT: L1, L2, L3

MODULE-3	**No. of Hours:8**

Normalization: Database Design Theory – Introduction to Normalization using Functional and Multivalued Dependencies: Informal design guidelines for relation schema, Functional Dependencies, Normal Forms basedon Primary Keys, Second and Third Normal Forms, Boyce-Codd Normal Form, Multivalued Dependency and Fourth Normal Form, Join Dependencies and Fifth Normal Form.
SQL: SQL data definition and data types, Schema change statements in SQL, specifying constraints in SQL,retrieval queries in SQL, INSERT, DELETE, and UPDATE statements in SQL, Additional features of SQL **Textbook 1: Ch 14.1 to 14.7, Ch 6.1 to 6.5**
RBT: L1, L2, L3

MODULE-4	**No. of Hours:8**

SQL: Advanced Queries: More complex SQL retrieval queries, Specifying constraints as assertions and action triggers, Views in SQL.

Transaction Processing: Introduction to Transaction Processing, Transaction and System concepts, Desirableproperties of Transactions, Characterizing schedules based on recoverability, Characterizing schedules based on Serializability, Transaction support in SQL.

Textbook 1: Ch 7.1 to 7.3, Ch 20.1 to 20.6RBT: L1, L2, L3

MODULE-5	**No. of Hours:08**

Concurrency Control in Databases: Two-phase locking techniques for Concurrency control, Concurrency control based on Timestamp ordering, Multiversion Concurrency control techniques, Validation Concurrency control techniques, Granularity of Data items and Multiple Granularity Locking.
NOSQL Databases and Big Data Storage Systems: Introduction to NOSQL Systems, The CAP Theorem, Document-Based NOSQL Systems and MongoDB, NOSQL Key-Value Stores, Column-Based or Wide Column NOSQL Systems, NOSQL Graph Databases and Neo4j

Textbook 1:Chapter 21.1 to 21.5, Chapter 24.1 to 24.6RBT: L1, L2, L3

Sl.NO	Experiments
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 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 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 Emp no 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 availablein 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 (Course Skill Set):

At the end of the course, the student will be able to:

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

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum

passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (maximum marks 50)

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
- 25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for **25 marks)**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.

CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voceand marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

SEE for IPCC

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course(**duration 03 hours**)

1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scoredby the student shall be proportionally scaled down to 50 Marks

The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIEcomponent only. Questions mentioned in the SEE paper may include questions from the practical component.

Suggested Learning Resources:

Text Books:

1. Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill

Activity Based Learning (Suggested Activities in Class)/ Practical Based learningMini Project:

- Project Based Learning

Experiment No-01

1. Create a table called Employee & execute the following.
Employee(EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)

```
CREATE TABLE Employee (
 EMPNO INT,
 ENAME VARCHAR(50),
 JOB VARCHAR(50),
 MANAGER_NO INT,
 SAL DECIMAL(10, 2),
 COMMISSION DECIMAL(10, 2));
```

1. **Create a user and grant all permissions to the user.**

   ```
   CREATE USER 'Abhi' IDENTIFIED BY 'abhi123';

   GRANT ALL PRIVILEGES ON * . * TO 'Abhi';
   ```

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.**

   ```
   INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
   VALUES (1, 'John Doe', 'Manager', NULL, 5000.00, 1000.00);

   INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
   VALUES (2, 'Jane Smith', 'Developer', 1, 4000.00, NULL);

   INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
   VALUES (3, 'Michael Johnson', 'Analyst', 1, 3500.00, 500.00);

   ROLLBACK;
   ```

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

   ```
   ALTER TABLE Employee ADD CONSTRAINT PK_Employee_EMPNO PRIMARY KEY (EMPNO);

   ALTER TABLE Employee MODIFY EMPNO INT NOT NULL;
   ```

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

   ```
   INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
   VALUES (4, NULL, NULL, NULL, NULL, NULL);

   SELECT * FROM Employee;
   ```

Experiment No-02

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

```
CREATE TABLE Employee1(
 EMPNO INT,
 ENAME VARCHAR(20),
 JOB VARCHAR(20),
 MGR INT,
 SAL DECIMAL(10, 2));
```

1. Add a column commission with domain to the Employee table

```
ALTER TABLE Employee
ADD commission DECIMAL(10,2);
```

2.Insert any five records into the table

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, commission)
VALUES
(101, 'Abhi', 'Manager', NULL, 50000.00, 100),
(102, 'Sunil', 'Salesperson', 101, 3000, 50),
(103, 'Rahul', 'Clerk', 101, 2000, 25),
(104, 'Kushal', 'Analyst', 101, 4000, 75),
(105, 'Prajwal', 'Salesperson', 102, 3500, 60);
```

3. Update the column details of job

```
UPDATE Employee
SET JOB = 'Senior Manager'
WHERE EMPNO = 101;
```

4. Rename the column of Employee table using alter command

```
ALTER TABLE Employee
RENAME COLUMN MGR TO MANAGER_ID;
```

5. Delete the employee whose Emp no is 105

```
DELETE FROM Employee
    WHERE EMPNO = 105;
```

Experiment No-03

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.**

```
CREATE TABLE Employee (
  E_id INT PRIMARY KEY,
  E_name VARCHAR(50),
  Age INT,
  Salary DECIMAL(10, 2));
```

2. **Count number of employee names from employee table**

```
SELECT COUNT(E_name) AS num_employees FROM Employee;
```

3. **Find the Maximum age from employee table**.

```
SELECT MAX(Age) AS max_age FROM Employee;
```

4. **Find the Minimum age from employee table.**

```
SELECT MIN(Age) AS min_age FROM Employee;
```

5. **Find salaries of employee in Ascending Order**

```
SELECT Salary FROM Employee ORDER BY Salary ASC;
```

6. **Find grouped salaries of employees**.

```
SELECT Salary, COUNT(*) AS num_employees_with_salary
FROM Employee
GROUP BY Salary;
```

Experiment No-04

Create a row level trigger for the customers table that would fire for INSERT or UPDATE orDELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.

CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)

CREATE A CUSTOMER TABLE

```
CREATE TABLE CUSTOMERS (
   ID INT PRIMARY KEY,
   NAME VARCHAR(50),
   AGE INT,
   ADDRESS VARCHAR(100),
   HOURLY_PAY DECIMAL(10, 2)
);
```

Inserting sample records into CUSTOMERS table

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, HOURLY_PAY) VALUES
(101, 'Abhi', 26, 'Hassan', 25.50),
(102, 'Jayanth', 28, 'Bengaluru', 28.75),
(103, 'Rahul', 35, 'Mandya', 30.00),
(104, 'Avinash', 25, 'Bengaluru', 22.00),
(105, 'Vivek', 40, 'Mysore', 32.50);
```

Add the salary column on Customer table

```
alter table customers add column salary decimal(10,2) after  hourly_pay;
```

set the salary

```
 update customers
   set salary = hourly_pay*11648;
```

- **Create trigger for before update salary**

```
create trigger before_houly_pay_update
   BEFORE UPDATE ON CUSTOMERS
    FOR EACH ROW
   SET NEW.SALARY =(NEW.HOURLY_PAY*11648);

 SHOW TRIGGERS;

select * from customers;
```

Increase The 50 Rupee per hour for Customer Id =101

```
 update customers
   set hourly_pay=50
   where id=101;
```

Increase 10 Rupee per hour for each and Every Customers

```
 update customers
 set hourly_pay=hourly_pay + 10;

select * from customers;
```

Create The Trigger for before insert salary

```
delete  from customers
    where id=105;

select * from customers;
```

```
create trigger before_houly_pay_insert
   BEFORE INSERT ON CUSTOMERS
    FOR EACH ROW
   SET NEW.SALARY =(NEW.HOURLY_PAY*11648);

 select * from customers;

INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, HOURLY_PAY, SALARY) VALUES
(105, 'Vivek', 22, 'Mysore', 25.50,NULL);

 select * from customers;
```

Create the trigger for after update the salary

```
 CREATE TABLE EXPENSES(
    EXPENSE_ID INT PRIMARY KEY,
    EXPENSE_NAME VARCHAR(50),
    EXPENSE_TOTAL DECIMAL(10,2)
    );

SELECT * FROM EXPENSES;

INSERT INTO EXPENSES
    VALUES (1,"salaries",0),
    (2,"supplies",0),
    (3,"taxes",0);

SELECT * FROM EXPENSES;

 update expenses
   set expense_total=(select sum(salary) from customers)
    where expense_name="salaries";

SELECT * FROM EXPENSES;
```

- **Create trigger for after delete salary**

```
 create trigger after_salary_delete
    AFTER DELETE ON CUSTOMERS
    FOR EACH ROW
   UPDATE EXPENSES
    SET EXPENSE_TOTAL=EXPENSE_TOTAL - OLD.SALARY
    WHERE EXPENSE_NAME = "salaries";

 SELECT * FROM EXPENSES;

delete from customers
    where id=5;

 SELECT * FROM EXPENSES;
```

- **Create trigger for after insert salary**

```
CREATE TRIGGER  after_salary_insert
    AFTER INSERT ON CUSTOMERS
    FOR EACH ROW
   UPDATE EXPENSES
    SET EXPENSE_TOTAL=EXPENSE_TOTAL + NEW.SALARY
    WHERE EXPENSE_NAME = "salaries";
```

```
 SELECT * FROM EXPENSES;

INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, HOURLY_PAY, SALARY) VALUES
(5, 'JAYANTH', 22, '123 Main St, Anytown, USA', 25.50,NULL);

SELECT * FROM EXPENSES;
```

- **Create trigger for after update salary**

```
CREATE TRIGGER after_salary_update
    AFTER UPDATE ON CUSTOMERS
    FOR EACH ROW
   UPDATE EXPENSES
    SET EXPENSE_TOTAL=EXPENSE_TOTAL + (NEW.SALARY - OLD.SALARY)
    WHERE EXPENSE_NAME = "salaries";

UPDATE CUSTOMERS
    SET hourly_pay =100
    WHERE ID=101;
```

Experiment No-05

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)

```
CREATE TABLE Employee (
   E_id INT,
   E_name VARCHAR(50),
   Age INT,
   Salary DECIMAL(10, 2)
);

INSERT INTO Employee (E_id, E_name, Age, Salary)
VALUES
   (101, 'Abhi', 26, 50000.00),
   (102, 'Jayanth', 28, 45000.00),
   (103, 'Sunil', 35, 60000.00);
DELIMITER //

CREATE PROCEDURE fetch_employee_data()
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE v_eid INT;
   DECLARE v_ename VARCHAR(50);
   DECLARE v_age INT;
   DECLARE v_salary DECIMAL(10, 2);

   -- Declare cursor for selecting data from Employee
   DECLARE cur CURSOR FOR
      SELECT E_id, E_name, Age, Salary
      FROM Employee;
   -- Declare continue handler to exit loop
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

   OPEN cur;

   read_loop: LOOP
      FETCH cur INTO v_eid, v_ename, v_age, v_salary;
      IF done THEN
         LEAVE read_loop;
      END IF;

      -- Process the fetched data (you can perform any operations here)
      SELECT CONCAT('Employee ID: ', v_eid, ', Name: ', v_ename, ', Age: ', v_age, ', Salary: ', v_salary) AS
employee_info;

   END LOOP;

   CLOSE cur;
END //

DELIMITER ;
```

To execute the stored procedure and see the results, you can call it like this:

```
CALL fetch_employee_data();
```

This will output the concatenated information for each employee fetched from the Employee table.

Experiment No-06

Write a PL/SQL block of code using parameterized Cursor, that will merge the data availablein 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;
```

1. Create the Tables

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

CREATE TABLE O_RollCall (
student_id INT PRIMARY KEY,
student_name VARCHAR(255),
birth_date DATE
);
```

2. Add Sample Records to both tables

```
INSERT INTO O_RollCall (student_id, student_name, birth_date) VALUES
 (1, 'Shivanna', '1995-08-15'), (3, 'Cheluva', '1990-12-10');

 INSERT INTO N_RollCall (student_id, student_name, birth_date)
VALUES   (1, 'Shivanna', '1995-08-15'),  (2, 'Bhadramma', '1998-03-22'), (3, 'Cheluva', '1990-12-10'),
 (4, 'Devendra', '2000-05-18'),  (5, 'Eshwar', '1997-09-03');
```

3. Define the Stored Procedure

```
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 n_cursor CURSOR FOR
SELECT student_id, student_name, birth_date
FROM N_RollCall;
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET done = TRUE;
OPEN n_cursor;
cursor_loop: LOOP
FETCH n_cursor INTO n_id, n_name, n_birth_date;
IF done THEN
LEAVE cursor_loop;
END IF;
IF NOT EXISTS (
SELECT 1
FROM O_RollCall
WHERE student_id = n_id
) THEN
INSERT INTO O_RollCall (student_id, student_name, birth_date)
VALUES (n_id, n_name, n_birth_date);
END IF;
END LOOP;
CLOSE n_cursor;
```

```
END//
DELIMITER ;
```

4.Execute the Stored Procedure

```
CALL merge_rollcall_data();
```

5.Verify Records in O_RollCall

```
SELECT * FROM O_RollCall;
```

Experiment No-07

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

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>
```

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

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 Programming books:

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

```
bookDB>
  title: "The Pragmatic Programmer: Your Journey to
  author: "David Thomas, Andrew
  category: "Software
  year:
})
```

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:
}
{
  _id:
  title: 'The Pragmatic Programmer: Your Journey to
  author: 'David Thomas, Andrew
  category: 'Software
  year:
}
]
```

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.updateOn
e(
  { title: "Clean Code: A Handbook of Agile Software Craftsmanship"
  },
```

```
  { $set: { author: "Robert C. Martin (Uncle Bob)" } }
)

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

b. Update Multiple Documents

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

```
bookDB>
  { year: { $lt: 2010 } },
  { $set: { category: "Classic Programming Books" } }
)

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

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

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
}
```

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
}
```

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
ProgrammingBook

bookDB>
tru

bookDB> show

bookDB>
```

2. Start MongoDB.

Launch the MongoDB daemon using the following command: