- SQL (Q1–Q7, Q28–Q32)
- PL/SQL (Q8–Q16, Q33–Q36)
- Python with SQL/MongoDB (Q17, Q18, Q26, Q27)
- MongoDB (Q19–Q25, Q37–Q39)

Q1 – Create and Query Student Table

```
-- 1(a) Create student table
CREATE TABLE student (
   roll_no INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(50),
   city VARCHAR(10),
   marks INT,
   email VARCHAR(50) UNIQUE,
   contact VARCHAR(15) NOT NULL
);

-- Insert 15 sample records
INSERT INTO student (name, city, marks, email, contact) VALUES
('Rohan', 'Pune', 85, 'rohan@gmail.com', '9876543210'),
('Amit', 'Mumbai', 72, 'amit@gmail.com', '9876543211'),
('Sneha', 'Delhi', 91, 'sneha@gmail.com', '9876543212'),
('Priya', 'Banglore', 66, 'priya@gmail.com', '9876543213'),
('Raj', 'Pune', 75, 'raj@gmail.com', '9876543214'),
('Kiran', 'Mumbai', 45, 'kiran@gmail.com', '9876543215'),
('Neha', 'Delhi', 83, 'neha@gmail.com', '9876543216'),
('Vikas', 'Banglore', 68, 'vikas@gmail.com', '9876543217'),
('Anita', 'Pune', 92, 'anita@gmail.com', '9876543218'),
('Ravi', 'Delhi', 58, 'ravi@gmail.com', '9876543219'),
('Meena', 'Mumbai', 77, 'meena@gmail.com', '9876543220'),
('Pooja', 'Banglore', 81, 'pooja@gmail.com', '9876543221'),
('Asha', 'Pune', 67, 'asha@gmail.com', '9876543222'),
('Sunil', 'Delhi', 49, 'sunil@gmail.com', '9876543223'),
('Deepak', 'Banglore', 79, 'deepak@gmail.com', '9876543224');

-- (b) Students from Pune with marks > 70
SELECT * FROM student WHERE city = 'Pune' AND marks > 70;

-- (c) Top 5 students with highest marks
SELECT * FROM student ORDER BY marks DESC LIMIT 5;

-- (d) Students whose marks > any student from Pune
SELECT * FROM student
WHERE marks > ANY (SELECT marks FROM student WHERE city = 'Pune');
```

Q2 – More Student Queries

```
-- (b) Students from Pune or Mumbai with marks between 50 and 80
SELECT * FROM student
WHERE (city = 'Pune' OR city = 'Mumbai') AND marks BETWEEN 50 AND 80;

-- (c) Average marks per city (only > 50)
SELECT city, AVG(marks) AS avg_marks
FROM student
GROUP BY city
HAVING AVG(marks) > 50;

-- (d) Students whose email is Gmail
SELECT * FROM student WHERE email LIKE '%@gmail.com';
```

Q3 – Views and Updates on Employees

```
-- Create table
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    salary DECIMAL(10,2)
);

-- Insert records
INSERT INTO employees VALUES (101, 'Rohan', 45000), (102, 'Amit', 50000), (103, 'Sneha', 60000);

-- Create view
CREATE VIEW emp_view AS SELECT emp_id, name, salary FROM employees;

-- (a) Increase salary by 1000 for emp_id = 101
UPDATE emp_view SET salary = salary + 1000 WHERE emp_id = 101;

-- (b) Add new employee
INSERT INTO emp_view VALUES (105, 'ABC', 50000);

-- Display all
SELECT * FROM employees;
```

Q4 – Subqueries (Customers & Orders)

```
-- (a) Create tables
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    product VARCHAR(50),
    amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);

-- (b) Insert sample records
INSERT INTO Customers VALUES
(1, 'Rohan', 'Pune'), (2, 'Amit', 'Delhi'), (3, 'Sneha', 'Mumbai'), (4, 'Priya', 'Pune'), (5, 'Raj', 'Delhi');

INSERT INTO Orders VALUES
(101, 1, 'Laptop', 55000), (102, 2, 'Mobile', 20000),
(103, 3, 'Tablet', 15000), (104, 4, 'TV', 30000), (105, 5, 'AC', 45000);

-- (c) Customer who spent highest in a single order
SELECT c.customer_name, o.amount
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.amount = (SELECT MAX(amount) FROM Orders);

-- (d) Customers with order amount > 100
SELECT c.customer_name, o.amount
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.amount > 100;
```

Q5 – Subquery with Total Order Amount

```
-- (c) Customers whose total order amount > 500
SELECT c.customer_name, SUM(o.amount) AS total
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name
```

```
HAVING SUM(o.amount) > 500;

-- (d) Subquery showing total order amount per customer
SELECT c.customer_name,
      (SELECT SUM(o.amount)
       FROM Orders o
       WHERE o.customer_id = c.customer_id) AS total_order_amount
FROM Customers c;
```

Q6 – Joins Example

```
-- (c) Customer names with product and amount
SELECT c.customer_name, o.product, o.amount
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id;

-- (d) All customers (even without orders)
SELECT c.customer_name, o.product, o.amount
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id;
```

Q7 – Joins with Missing Data

```
-- (c) All orders even if customer info missing
SELECT o.order_id, c.customer_name, o.product, o.amount
FROM Orders o
LEFT JOIN Customers c ON o.customer_id = c.customer_id;

-- (d) All customers and all orders
SELECT c.customer_name, o.product, o.amount
FROM Customers c
FULL OUTER JOIN Orders o ON c.customer_id = o.customer_id;
```

Q28 – Multiple Table Joins

```
-- Table structures
CREATE TABLE Customer (cust_no INT PRIMARY KEY, fname VARCHAR(30), lname VARCHAR(30));
CREATE TABLE Cust_Acc_FD_Details (custno INT, acc_fd_no INT);
CREATE TABLE fd_details (acc_fd_no INT, amt DECIMAL(10,2));
CREATE TABLE address_details (code_no INT PRIMARY KEY, add1 VARCHAR(50), add2 VARCHAR(50), state VARCHAR(30), city
VARCHAR(30), pincode VARCHAR(10));
CREATE TABLE Employee (emp_no INT PRIMARY KEY, f_name VARCHAR(30), l_name VARCHAR(30), m_name VARCHAR(30), dept
VARCHAR(30), code_no INT);
CREATE TABLE Contact_details (code_no INT, cntc_type VARCHAR(20), cntc_data VARCHAR(50));

-- (a) Address of specific customer
SELECT * FROM address_details
WHERE code_no IN (
  SELECT code_no FROM Employee
  WHERE f_name='xyz' AND l_name='pqr'
);

-- (b) Customer holding FD > 5000
SELECT c.fname, c.lname, f.amt
FROM Customer c
JOIN Cust_Acc_FD_Details ca ON c.cust_no = ca.custno
JOIN fd_details f ON ca.acc_fd_no = f.acc_fd_no
WHERE f.amt > 5000;

-- (c) Employee details with contact details (LEFT & RIGHT JOIN)
SELECT e.*, cd.cntc_type, cd.cntc_data
FROM Employee e
LEFT JOIN Contact_details cd ON e.code_no = cd.code_no;

SELECT e.*, cd.cntc_type, cd.cntc_data
FROM Employee e
RIGHT JOIN Contact_details cd ON e.code_no = cd.code_no;
```

Q29–Q32 – Bank Database Queries

```
-- Common Schema
CREATE TABLE Account(Acc_no INT PRIMARY KEY, branch_name VARCHAR(50), balance DECIMAL(10,2));
CREATE TABLE Branch(branch_name VARCHAR(50) PRIMARY KEY, branch_city VARCHAR(50), assets DECIMAL(10,2));
CREATE TABLE Customer(cust_name VARCHAR(50), cust_street VARCHAR(50), cust_city VARCHAR(50));
CREATE TABLE Depositor(cust_name VARCHAR(50), acc_no INT);
CREATE TABLE Loan(loan_no INT PRIMARY KEY, branch_name VARCHAR(50), amount DECIMAL(10,2));
CREATE TABLE Borrower(cust_name VARCHAR(50), loan_no INT);

-- Q29(a) View on borrower (two columns)
CREATE VIEW borrower_view AS SELECT cust_name, loan_no FROM Borrower;
INSERT INTO borrower_view VALUES ('Rohan', 201);
UPDATE borrower_view SET cust_name = 'Amit' WHERE loan_no = 201;
DELETE FROM borrower_view WHERE cust_name = 'Amit';

-- Q29(b) View on borrower & depositor
CREATE VIEW bd_view AS
SELECT b.cust_name, d.acc_no
FROM Borrower b, Depositor d
WHERE b.cust_name = d.cust_name;
INSERT INTO bd_view VALUES ('Sneha', 1001);

-- Q29(c) Customers without bank branches in Akurdi
SELECT cust_name
FROM Customer
WHERE cust_city <> 'Akurdi';

-- Q29(d) Customers with loan accounts
SELECT DISTINCT c.cust_name
FROM Customer c
JOIN Borrower b ON c.cust_name = b.cust_name;

-- Q30(a–e)
SELECT DISTINCT branch_name FROM Loan;
SELECT loan_no FROM Loan WHERE branch_name='Akurdi' AND amount>12000;
SELECT c.cust_name, b.loan_no, l.amount
FROM Customer c
JOIN Borrower b ON c.cust_name=b.cust_name
JOIN Loan l ON b.loan_no=l.loan_no;
SELECT c.cust_name FROM Customer c
JOIN Borrower b ON c.cust_name=b.cust_name
JOIN Loan l ON b.loan_no=l.loan_no
WHERE l.branch_name='Akurdi'
ORDER BY c.cust_name;
SELECT cust_name FROM Customer
WHERE cust_name IN (
   SELECT cust_name FROM Depositor
   UNION
   SELECT cust_name FROM Borrower
);

-- Q31(a–f)
SELECT branch_name, AVG(balance) AS avg_bal FROM Account GROUP BY branch_name;
SELECT branch_name, COUNT(DISTINCT acc_no) AS num_depositors FROM Depositor NATURAL JOIN Account GROUP BY branch_name;
SELECT branch_name FROM Account GROUP BY branch_name HAVING AVG(balance)>12000;
SELECT COUNT(*) AS num_customers FROM Customer;
SELECT SUM(amount) AS total_loan FROM Loan;
DELETE FROM Loan WHERE amount BETWEEN 1300 AND 1500;
DELETE FROM Branch WHERE branch_city='Nigdi';

-- Q32(a–e)
SELECT DISTINCT c.cust_name
FROM Customer c
JOIN Depositor d ON c.cust_name=d.cust_name
JOIN Borrower b ON c.cust_name=b.cust_name;

SELECT DISTINCT c.cust_name
```

```
FROM Customer c
WHERE c.cust_name IN (SELECT cust_name FROM Depositor)
AND c.cust_name NOT IN (SELECT cust_name FROM Borrower);

SELECT AVG(balance) FROM Account WHERE branch_name='Akurdi';
SELECT branch_name, COUNT(acc_no) FROM Depositor NATURAL JOIN Account GROUP BY branch_name;
SELECT branch_name FROM Account GROUP BY branch_name HAVING AVG(balance)>12000;
```

Q8 – Library Fine Calculation Procedure

```
-- Create tables
CREATE TABLE Borrower (
  Roll_no INT,
  Name VARCHAR2(30),
  DateofIssue DATE,
  NameofBook VARCHAR2(50),
  Status CHAR(1)
);

CREATE TABLE Fine (
  Roll_no INT,
  Date_ DATE,
  Amt NUMBER
);

-- Procedure to calculate fine
CREATE OR REPLACE PROCEDURE calc_fine (
  p_rollno IN INT,
  p_bookname IN VARCHAR2
) IS
  v_dateofissue DATE;
  v_days NUMBER;
  v_fine NUMBER;
BEGIN
  SELECT DateofIssue INTO v_dateofissue
  FROM Borrower
  WHERE Roll_no = p_rollno AND NameofBook = p_bookname;

  v_days := SYSDATE - v_dateofissue;

  IF v_days BETWEEN 15 AND 30 THEN
     v_fine := v_days * 5;
  ELSIF v_days > 30 THEN
     v_fine := v_days * 50;
  ELSE
     v_fine := 0;
  END IF;

  UPDATE Borrower
  SET Status = 'R'
  WHERE Roll_no = p_rollno AND NameofBook = p_bookname;

  IF v_fine > 0 THEN
     INSERT INTO Fine VALUES (p_rollno, SYSDATE, v_fine);
  END IF;
END;
/

-- Execute
BEGIN
  calc_fine(101, 'DBMS Concepts');
END;
/
```

Q9 – PL/SQL Procedure to Calculate Area of Circle

```
CREATE TABLE areas (
  radius NUMBER,
  area NUMBER
);

CREATE OR REPLACE PROCEDURE circle_area IS
  r NUMBER;
  a NUMBER;
BEGIN
  FOR r IN 5..9 LOOP
    a := 3.14159 * r * r;
    INSERT INTO areas VALUES (r, a);
  END LOOP;
END;
/

-- Execute
BEGIN
  circle_area;
END;
/
```

Q10 – Stored Function: Get Employee Bonus

```
CREATE TABLE employees (
  emp_id INT PRIMARY KEY,
  emp_name VARCHAR2(30),
  salary NUMBER
);

INSERT INTO employees VALUES (1, 'Rohan', 40000);

CREATE OR REPLACE FUNCTION get_emp_bonus(p_emp_id INT)
RETURN NUMBER IS
  v_salary NUMBER;
  v_bonus NUMBER;
BEGIN
  SELECT salary INTO v_salary FROM employees WHERE emp_id = p_emp_id;
  v_bonus := v_salary * 0.15;
  RETURN v_bonus;
END;
/

-- Call function
DECLARE
  v_b NUMBER;
BEGIN
  v_b := get_emp_bonus(1);
  DBMS_OUTPUT.PUT_LINE('Bonus: ' || v_b);
END;
/
```

Q11 – Stored Function: Get Student Grade

```
CREATE TABLE students (
  student_id INT PRIMARY KEY,
  student_name VARCHAR2(30),
  marks INT
);

INSERT INTO students VALUES (101, 'Amit', 85);

CREATE OR REPLACE FUNCTION get_student_grade(p_id INT)
RETURN VARCHAR2 IS
  v_marks INT;
  v_grade VARCHAR2(20);
```

```
BEGIN
  SELECT marks INTO v_marks FROM students WHERE student_id = p_id;

  IF v_marks BETWEEN 76 AND 100 THEN
      v_grade := 'Distinction';
  ELSIF v_marks BETWEEN 61 AND 75 THEN
      v_grade := 'First Class';
  ELSIF v_marks BETWEEN 41 AND 60 THEN
      v_grade := 'Higher Second Class';
  ELSE
      v_grade := 'Fail';
  END IF;

  RETURN v_grade;
END;
/

-- Call function
DECLARE
  g VARCHAR2(30);
BEGIN
  g := get_student_grade(101);
  DBMS_OUTPUT.PUT_LINE('Grade: ' || g);
END;
/
```

Q12 – Merge Data Using Explicit Cursor

```
CREATE TABLE O_RollCall (Rollno NUMBER, Name VARCHAR2(30));
CREATE TABLE N_RollCall (Rollno NUMBER, Name VARCHAR2(30));

CREATE OR REPLACE PROCEDURE merge_rollcall IS
  CURSOR c1 IS SELECT * FROM N_RollCall;
  v_count NUMBER;
BEGIN
  FOR rec IN c1 LOOP
    SELECT COUNT(*) INTO v_count FROM O_RollCall WHERE Rollno = rec.Rollno;
    IF v_count = 0 THEN
      INSERT INTO O_RollCall VALUES (rec.Rollno, rec.Name);
    END IF;
  END LOOP;
END;
/

-- Execute
BEGIN
  merge_rollcall;
END;
/
```

Q13 – Parameterized Cursor

```
CREATE OR REPLACE PROCEDURE merge_rollcall_param(p_rollno NUMBER) IS
  CURSOR c1 IS SELECT * FROM N_RollCall WHERE Rollno > p_rollno;
  v_count NUMBER;
BEGIN
  FOR rec IN c1 LOOP
    SELECT COUNT(*) INTO v_count FROM O_RollCall WHERE Rollno = rec.Rollno;
    IF v_count = 0 THEN
      INSERT INTO O_RollCall VALUES (rec.Rollno, rec.Name);
    END IF;
  END LOOP;
END;
/

-- Execute
BEGIN
  merge_rollcall_param(5);
```

```
END;
/
```

Q14 – Implicit Cursor Example

```
CREATE OR REPLACE PROCEDURE merge_implicit IS
BEGIN
  INSERT INTO O_RollCall(Rollno, Name)
  SELECT n.Rollno, n.Name FROM N_RollCall n
  WHERE n.Rollno NOT IN (SELECT Rollno FROM O_RollCall);
END;
/

-- Execute
BEGIN
  merge_implicit;
END;
/
```

Q15 – Trigger: Insert or Update Audit

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR2(30),
  salary NUMBER
);

CREATE TABLE emp_audit (
  id INT,
  event_type VARCHAR2(30),
  event_date DATE
);

-- Trigger
CREATE OR REPLACE TRIGGER emp_audit_trg
AFTER INSERT OR BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO emp_audit VALUES(:NEW.id, 'INSERT', SYSDATE);
  ELSIF UPDATING THEN
    INSERT INTO emp_audit VALUES(:OLD.id, 'UPDATE', SYSDATE);
  END IF;
END;
/
```

Q16 – Trigger: Delete or Update Audit

```
CREATE OR REPLACE TRIGGER emp_audit_trg2
AFTER DELETE OR BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
  IF DELETING THEN
    INSERT INTO emp_audit VALUES(:OLD.id, 'DELETE', SYSDATE);
  ELSIF UPDATING THEN
    INSERT INTO emp_audit VALUES(:OLD.id, 'UPDATE', SYSDATE);
  END IF;
END;
/
```

Q33 – Library Fine with PL/SQL (Similar to Q8)

```
CREATE OR REPLACE PROCEDURE proc_fine (
  p_rollno IN INT,
  p_bookname IN VARCHAR2
) IS
  v_date DATE;
  v_days NUMBER;
  v_fine NUMBER;
BEGIN
```

```
  SELECT DateofIssue INTO v_date
  FROM Borrower
  WHERE Roll_no = p_rollno AND NameofBook = p_bookname;

  v_days := SYSDATE - v_date;

  IF v_days BETWEEN 15 AND 30 THEN
    v_fine := v_days * 5;
  ELSIF v_days > 30 THEN
    v_fine := v_days * 50;
  ELSE
    v_fine := 0;
  END IF;

  UPDATE Borrower SET Status = 'R'
  WHERE Roll_no = p_rollno AND NameofBook = p_bookname;

  IF v_fine > 0 THEN
    INSERT INTO Fine VALUES (p_rollno, SYSDATE, v_fine);
  END IF;
END;
/
```

Q34 – Parameterized Cursor (Again, Merge Tables)

```
CREATE TABLE O_RollCall (Rollno NUMBER, Name VARCHAR2(30));
CREATE TABLE N_RollCall (Rollno NUMBER, Name VARCHAR2(30));

CREATE OR REPLACE PROCEDURE merge_param(p_rollno NUMBER) IS
  CURSOR c1 IS SELECT * FROM N_RollCall WHERE Rollno > p_rollno;
  v_count NUMBER;
BEGIN
  FOR rec IN c1 LOOP
    SELECT COUNT(*) INTO v_count FROM O_RollCall WHERE Rollno = rec.Rollno;
    IF v_count = 0 THEN
      INSERT INTO O_RollCall VALUES (rec.Rollno, rec.Name);
    END IF;
  END LOOP;
END;
/
```

Q35 – Procedure for Student Grade Categorization

```
CREATE TABLE stud_marks (
  rollno NUMBER(10),
  name VARCHAR2(30),
  total_marks NUMBER(10)
);

CREATE TABLE Result (
  rollno NUMBER(10),
  name VARCHAR2(30),
  class VARCHAR2(30)
);

CREATE OR REPLACE PROCEDURE proc_Grade IS
  CURSOR c1 IS SELECT * FROM stud_marks;
  v_class VARCHAR2(30);
BEGIN
  FOR rec IN c1 LOOP
    IF rec.total_marks BETWEEN 990 AND 1500 THEN
        v_class := 'Distinction';
    ELSIF rec.total_marks BETWEEN 900 AND 989 THEN
        v_class := 'First Class';
    ELSIF rec.total_marks BETWEEN 825 AND 899 THEN
        v_class := 'Higher Second Class';
    ELSIF rec.total_marks BETWEEN 600 AND 824 THEN
        v_class := 'Pass Class';
    ELSE
        v_class := 'Fail';
    END IF;
    INSERT INTO Result VALUES (rec.rollno, rec.name, v_class);
  END LOOP;
END;
/

-- Execute
BEGIN
  proc_Grade;
END;
/
```

Q36 – Trigger for Library Audit

```
CREATE TABLE Library (
  bookid NUMBER,
  bookname VARCHAR2(30),
  issuedate DATE,
  returndate DATE,
  cardnumber VARCHAR2(10)
);

CREATE TABLE Lib_Audit (
  bookid NUMBER,
  bookname VARCHAR2(30),
  issuedate DATE,
  returndate DATE,
  cardnumber VARCHAR2(10)
);

CREATE OR REPLACE TRIGGER trg_Library_Audit
AFTER DELETE OR UPDATE ON Library
FOR EACH ROW
BEGIN
  INSERT INTO Lib_Audit
  VALUES(:OLD.bookid, :OLD.bookname, :OLD.issuedate, :OLD.returndate, :OLD.cardnumber);
END;
/
```

Q17 – Library Table (MySQL/Oracle Connectivity)

```
# Program: Create Library table, insert 5 records, delete 2 records
import mysql.connector

# Connect to MySQL
con = mysql.connector.connect(
    host="localhost",
    user="root",
    password="your_password",
    database="collegeDB"
)

cur = con.cursor()

# Create table
cur.execute("""
CREATE TABLE IF NOT EXISTS library (
    id INT PRIMARY KEY,
    bookname VARCHAR(50),
    authorname VARCHAR(50)
)
""")
```

```
# Insert 5 records
books = [
    (1, 'DBMS Concepts', 'Korth'),
    (2, 'Let Us C', 'Yashwant Kanetkar'),
    (3, 'Operating Systems', 'Galvin'),
    (4, 'Computer Networks', 'Tanenbaum'),
    (5, 'Data Structures', 'Lipschutz')
]
cur.executemany("INSERT INTO library VALUES (%s, %s, %s)", books)
con.commit()

# Delete 2 records
cur.execute("DELETE FROM library WHERE id IN (4,5)")
con.commit()

print("Records inserted and deleted successfully.")
con.close()
```

Q18 – Vehicles Table (MySQL/Oracle Connectivity)

```
# Program: Create Vehicles table, insert 5 records, update 2 records
import mysql.connector

con = mysql.connector.connect(
    host="localhost",
    user="root",
    password="your_password",
    database="collegeDB"
)
cur = con.cursor()

# Create table
cur.execute("""
CREATE TABLE IF NOT EXISTS vehicles (
    id INT PRIMARY KEY,
    vehicle_name VARCHAR(50),
    price DECIMAL(10,2)
)
""")

# Insert 5 records
vehicles = [
    (1, 'Car', 800000),
    (2, 'Bike', 120000),
    (3, 'Truck', 1500000),
    (4, 'Scooter', 90000),
    (5, 'Bus', 2000000)
]
cur.executemany("INSERT INTO vehicles VALUES (%s, %s, %s)", vehicles)
con.commit()

# Update 2 records
cur.execute("UPDATE vehicles SET price = price + 50000 WHERE id IN (1,3)")
con.commit()

print("Records inserted and updated successfully.")
con.close()
```

Q26 – MongoDB Connectivity: Students Collection (CRUD)

```
# Program: MongoDB CRUD operations for students collection
from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["collegeDB"]
students = db["students"]
```

```
# Create (Insert)
students.insert_many([
    {"roll": 1, "name": "Rohan", "marks": 88},
    {"roll": 2, "name": "Sneha", "marks": 75},
    {"roll": 3, "name": "Amit", "marks": 60}
])
print("Inserted sample records.")

# Read
for s in students.find():
    print(s)

# Update
students.update_one({"roll": 2}, {"$set": {"marks": 82}})
print("Updated student marks.")

# Delete
students.delete_one({"roll": 3})
print("Deleted one student record.")

client.close()
```

Q27 – MongoDB Connectivity: Departments Collection (CRUD)

```
# Program: MongoDB CRUD operations for departments collection
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["collegeDB"]
departments = db["departments"]

# Create (Insert)
departments.insert_many([
    {"id": 1, "name": "Computer", "hod": "Dr. Mehta", "studentcount": 120},
    {"id": 2, "name": "Mechanical", "hod": "Dr. Patil", "studentcount": 100},
    {"id": 3, "name": "ENTC", "hod": "Dr. Kulkarni", "studentcount": 80}
])
print("Inserted departments.")

# Read
for d in departments.find():
    print(d)

# Update
departments.update_one({"name": "ENTC"}, {"$set": {"studentcount": 90}})
print("Updated student count for ENTC.")

# Delete
departments.delete_one({"name": "Mechanical"})
print("Deleted one department record.")

client.close()
```

Q19 – Library Collection (Using Logical Operators)

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017/")
db = client["collegeDB"]
library = db["Library"]

# (a) Insert 5 documents
library.insert_many([
    {"bookname": "Rich Dad Poor Dad", "authorname": "Kiyosaki", "genre": "Business"},
    {"bookname": "Harry Potter", "authorname": "Rowling", "genre": "Fiction"},
    {"bookname": "Atomic Habits", "authorname": "James Clear", "genre": "Self-Help"},
    {"bookname": "Zero to One", "authorname": "Peter Thiel", "genre": "Business"},
    {"bookname": "Lord of the Rings", "authorname": "Tolkien", "genre": "Fantasy"}
])

# (i) Genre = Fiction OR Business
for doc in library.find({"$or": [{"genre": "Fiction"}, {"genre": "Business"}]}):
    print(doc)

# (ii) Books NOT written by "Rowling"
for doc in library.find({"authorname": {"$ne": "Rowling"}}):
    print(doc)

# (iii) Books by specific author and genre
for doc in library.find({"authorname": "Peter Thiel", "genre": "Business"}):
    print(doc)
```

Q20 – Library Collection (Sort & Update)

```
# (b) Sort and fetch 3 books in descending order of genre
for doc in library.find().sort("genre", -1).limit(3):
    print(doc)

# (c) Update genre of specific book
library.update_one({"bookname": "Atomic Habits"}, {"$set": {"genre": "Motivation"}})

# (d) Delete books of a specific genre
library.delete_many({"genre": "Fantasy"})
```

Q21 – Library Collection (Update by Author)

```
# (b) Sort and fetch 3 books descending by bookname
for doc in library.find().sort("bookname", -1).limit(3):
    print(doc)

# (c) Update genre for all books by certain author
library.update_many({"authorname": "Kiyosaki"}, {"$set": {"genre": "Finance"}})

# (d) Delete a book by name
library.delete_one({"bookname": "Zero to One"})
```

Q22 – Orders Collection (Aggregation & Indexing)

```
orders = db["Orders"]
orders.insert_many([
    {"order_id": 1, "customername": "Rohan", "product": "Laptop", "amount": 50000, "city": "Pune"},
    {"order_id": 2, "customername": "Amit", "product": "Mobile", "amount": 20000, "city": "Delhi"},
    {"order_id": 3, "customername": "Sneha", "product": "TV", "amount": 30000, "city": "Mumbai"},
    {"order_id": 4, "customername": "Raj", "product": "Tablet", "amount": 15000, "city": "Pune"},
    {"order_id": 5, "customername": "Priya", "product": "AC", "amount": 40000, "city": "Delhi"}
])

# (b) Total sales amount across all orders
pipeline = [{"$group": {"_id": None, "total_sales": {"$sum": "$amount"}}}]
print(list(orders.aggregate(pipeline)))

# (c) Create index on customer and amount
orders.create_index([("customername", 1), ("amount", 1)])

# (d) Show all indexes
print(orders.index_information())
```

Q23 – Unique Index & Aggregation Per Product

```
# (b) Create unique index on customer field
orders.create_index("customername", unique=True)

# (c) Average order amount per product
```

```
pipeline = [{"$group": {"_id": "$product", "avg_amount": {"$avg": "$amount"}}}]
for res in orders.aggregate(pipeline):
    print(res)

# (d) Show all indexes
print(orders.index_information())
```

Q24 – Books Collection (MapReduce)

```
books = db["books"]
books.insert_many([
    {"name": "Book1", "genre": "Fiction"},
    {"name": "Book2", "genre": "Fiction"},
    {"name": "Book3", "genre": "Science"},
    {"name": "Book4", "genre": "Fiction"},
    {"name": "Book5", "genre": "Science"},
    {"name": "Book6", "genre": "History"},
    {"name": "Book7", "genre": "Science"},
    {"name": "Book8", "genre": "History"},
    {"name": "Book9", "genre": "Fiction"},
    {"name": "Book10", "genre": "Science"}
])

# (b) Map and Reduce functions
mapf = """function() { emit(this.genre, 1); }"""
reducef = """function(key, values) { return Array.sum(values); }"""

result = books.map_reduce(mapf, reducef, "genre_summary")

# (c) Display results
for doc in result.find():
    print(doc)
```

Q25 – Sales Collection (MapReduce with Sum)

```
sales = db["sales"]
sales.insert_many([
    {"item": "Laptop", "category": "Electronics", "quantity": 5},
    {"item": "TV", "category": "Electronics", "quantity": 3},
    {"item": "Book", "category": "Stationary", "quantity": 10},
    {"item": "Pen", "category": "Stationary", "quantity": 20},
    {"item": "Shoes", "category": "Fashion", "quantity": 6}
])

# (b) Map function
mapf = """function() { emit(this.category, this.quantity); }"""

# (c) Reduce function
reducef = """function(key, values) { return Array.sum(values); }"""

# (d) Execute MapReduce and store in new collection
result = sales.map_reduce(mapf, reducef, "summary_collection")

for doc in result.find():
    print(doc)
```

Q37 – Aggregation and Unique Index

```
students = db["Student_Data"]
students.insert_many([
    {"RollNo": 1, "Name": "Rohan", "Class": "TE", "dept": "Comp", "city": "Pune", "Marks": 88, "Mobile": "1111"},
    {"RollNo": 2, "Name": "Amit", "Class": "BE", "dept": "ENTC", "city": "Mumbai", "Marks": 75, "Mobile": "2222"},
    {"RollNo": 3, "Name": "Sneha", "Class": "TE", "dept": "Comp", "city": "Pune", "Marks": 91, "Mobile": "3333"}
])

# (a) Department-wise Max Marks
pipeline = [{"$group": {"_id": "$dept", "MaxMarks": {"$max": "$Marks"}}}]
print(list(students.aggregate(pipeline)))
```

```
# (b) Class-wise Min Marks for Pune city
pipeline = [{"$match": {"city": "Pune"}}, {"$group": {"_id": "$Class", "MinMarks": {"$min": "$Marks"}}}]
print(list(students.aggregate(pipeline)))

# (c) City-wise Sum of Marks
pipeline = [{"$group": {"_id": "$city", "TotalMarks": {"$sum": "$Marks"}}}]
print(list(students.aggregate(pipeline)))

# (d) Unique index on Mobile
students.create_index("Mobile", unique=True)
```

Q38 – MongoDB CRUD and Logical Operators

```
stud = db["Student_Data_CRUD"]
stud.insert_many([
    {"RollNo": 1, "Name": "Rohan", "Class": "TE", "dept": "Comp", "City": "Pune", "Marks": 78, "mobile": "1111", "sport_status": "Y"},
    {"RollNo": 2, "Name": "Amit", "Class": "BE", "dept": "IT", "City": "Mumbai", "Marks": 88, "mobile": "2222", "sport_status": "N"},
    {"RollNo": 3, "Name": "Sneha", "Class": "TE", "dept": "Comp", "City": "Pune", "Marks": 92, "mobile": "3333", "sport_status": "Y"},
    {"RollNo": 4, "Name": "Neha", "Class": "TE", "dept": "ENTC", "City": "Pune", "Marks": 35, "mobile": "4444", "sport_status": "N"}
])

# (b) Update mobile number
stud.update_one({"Name": "Rohan"}, {"$set": {"mobile": "9999"}})

# (c) Update class TE→BE
stud.update_many({"Class": "TE"}, {"$set": {"Class": "BE"}})

# (d) Delete students with Marks < 40
stud.delete_many({"Marks": {"$lt": 40}})

# (e) Department-wise first 3 toppers
pipeline = [
    {"$sort": {"Marks": -1}},
    {"$group": {"_id": "$dept", "toppers": {"$push": "$Name"}}},
    {"$project": {"toppers": {"$slice": ["$toppers", 3]}}}
]
print(list(stud.aggregate(pipeline)))

# (f) First class (≥60) and sports participants
for s in stud.find({"$and": [{"Marks": {"$gte": 60}}, {"sport_status": "Y"}]}):
    print(s)

# (g) First class OR sports participants
for s in stud.find({"$or": [{"Marks": {"$gte": 60}}, {"sport_status": "Y"}]}):
    print(s)
```

Q39 – Employee Collection with Aggregation and CRUD

```
emp = db["Employee"]
emp.insert_many([
    {"emp_id": 1, "emp_name": "Rohan", "emp_salary": 45000, "emp_designation": "Junior Developer", "emp_Bdate": "1989-05-10",
"emp_mobileno": "9999", "emp_dept": "IT"},
    {"emp_id": 2, "emp_name": "Amit", "emp_salary": 55000, "emp_designation": "Senior Developer", "emp_Bdate": "1985-08-15",
"emp_mobileno": "8888", "emp_dept": "Comp"},
    {"emp_id": 3, "emp_name": "Sneha", "emp_salary": 60000, "emp_designation": "Junior Developer", "emp_Bdate": "1960-04-12",
"emp_mobileno": "7777", "emp_dept": "IT"}
])

# (b) Total salary department-wise
pipeline = [{"$group": {"_id": "$emp_dept", "TotalSalary": {"$sum": "$emp_salary"}}}]
print(list(emp.aggregate(pipeline)))

# (c) Update designation
emp.update_many({"emp_designation": "Junior Developer"}, {"$set": {"emp_designation": "Senior Developer"}})

# (d) Delete employees age > 60 (approx DOB before 1965)
emp.delete_many({"emp_Bdate": {"$lt": "1965-01-01"}})
```

```
# (e) Display names and salary department-wise
pipeline = [{"$group": {"_id": "$emp_dept", "Employees": {"$push": {"name": "$emp_name", "salary": "$emp_salary"}}}}]
print(list(emp.aggregate(pipeline)))
```