(1)

```
-- 1. Branch names from Loan
SELECT DISTINCT branch_name FROM Loan;

-- 2. Loan numbers at Wadia College branch with amount > 12000
SELECT loan_no FROM Loan
WHERE branch_name = 'Wadia College' AND amount > 12000;

-- 3. Customers with their loan number and amount
SELECT Borrower.cust_name, Loan.loan_no, Loan.amount
FROM Borrower, Loan
WHERE Borrower.loan_no = Loan.loan_no;

-- 4. Customers having loan from Wadia College branch (A–Z)
SELECT Borrower.cust_name
FROM Borrower, Loan
WHERE Borrower.loan_no = Loan.loan_no
AND Loan.branch_name = 'Wadia College'
ORDER BY Borrower.cust_name;

-- 5. Unique branch cities
SELECT DISTINCT branch_city FROM Branch;
```

(2)

```
-- 1. Customers who have both account and loan
SELECT DISTINCT Depositor.cust_name
FROM Depositor, Borrower
WHERE Depositor.cust_name = Borrower.cust_name;

-- 2. Customers who have account or loan or both
SELECT cust_name FROM Depositor
UNION
SELECT cust_name FROM Borrower;

-- 3. Customers who have account but no loan
SELECT DISTINCT Depositor.cust_name
FROM Depositor
WHERE Depositor.cust_name NOT IN (SELECT cust_name FROM Borrower);

-- 4. Average account balance at 'Wadia College' branch
SELECT AVG(balance) AS avg_balance
FROM Account
WHERE branch_name = 'Wadia College';

-- 5. Number of depositors at each branch
SELECT A.branch_name, COUNT(D.cust_name) AS no_of_depositors
FROM Account A, Depositor D
WHERE A.acc_no = D.acc_no
GROUP BY A.branch_name;
```

(3)

```
-- 1. Branches where average account balance > 15000
SELECT branch_name
FROM Account
GROUP BY branch_name
HAVING AVG(balance) > 15000;

-- 2. Number of tuples in Customer relation
SELECT COUNT(*) AS total_customers
FROM Customer;

-- 3. Total loan amount given by bank
SELECT SUM(amount) AS total_loan_amount
FROM Loan;

-- 4. Delete all loans with amount between 1300 and 1500
DELETE FROM Loan
WHERE amount BETWEEN 1300 AND 1500;

-- 5. Average account balance at each branch
SELECT branch_name, AVG(balance) AS avg_balance
FROM Account
GROUP BY branch_name;

-- 6. Customer name and city where name starts with 'P'
SELECT cust_name, cust_city
FROM Customer
WHERE cust_name LIKE 'P%';
```

(4)

```
--List names of customers having 'A' as the second letter in their name
SELECT Cust_name
FROM Cust_Master
```

```
WHERE Cust_name LIKE '_A%';

--Display orders from customers no C1002, C1005, C1007, and C1008
SELECT *
FROM Orders
WHERE Cust_no IN ('C1002','C1005','C1007','C1008');

--List clients who stay in either 'Banglore' or 'Manglore'
SELECT *
FROM Cust_Master
WHERE Cust_addr IN ('Banglore','Manglore');

--Display customer names and the product names they purchased
SELECT Cust_Master.Cust_name, Product.Product_name
FROM Cust_Master, Orders, Product
WHERE Cust_Master.Cust_no = Orders.Cust_no
AND Orders.Order_no = Product.Order_no;

--Create a view View1 consisting of Cust_name and Product_name
CREATE VIEW View1 AS
SELECT Cust_Master.Cust_name, Product.Product_name
FROM Cust_Master, Orders, Product
WHERE Cust_Master.Cust_no = Orders.Cust_no
AND Orders.Order_no = Product.Order_no;

--Display product name and quantity purchased by each customer
SELECT Cust_Master.Cust_name, Product.Product_name, Orders.Qty_Ordered
FROM Cust_Master, Orders, Product
WHERE Cust_Master.Cust_no = Orders.Cust_no
AND Orders.Order_no = Product.Order_no;

--Perform different join operations (examples)
--Inner Join
SELECT Cust_Master.Cust_name, Product.Product_name
FROM Cust_Master
INNER JOIN Orders ON Cust_Master.Cust_no = Orders.Cust_no
INNER JOIN Product ON Orders.Order_no = Product.Order_no;

--Left Join
SELECT Cust_Master.Cust_name, Product.Product_name
FROM Cust_Master
LEFT JOIN Orders ON Cust_Master.Cust_no = Orders.Cust_no
LEFT JOIN Product ON Orders.Order_no = Product.Order_no;

--Right Join
SELECT Cust_Master.Cust_name, Product.Product_name
FROM Cust_Master
RIGHT JOIN Orders ON Cust_Master.Cust_no = Orders.Cust_no
RIGHT JOIN Product ON Orders.Order_no = Product.Order_no;

(5)
-- 1. Find the names of all employees who work for 'TCS'
SELECT employee_name
FROM Works
WHERE company_name = 'TCS';

-- 2. Find names and company names of all employees sorted by company name (ASC) and employee name (DESC)
SELECT employee_name, company_name
FROM Works
ORDER BY company_name ASC, employee_name DESC;

-- 3. Change city of employees working with 'InfoSys' to 'Bangalore'
UPDATE Employee
SET city = 'Bangalore'
WHERE employee_name IN (
   SELECT employee_name FROM Works WHERE company_name = 'InfoSys'
);

-- 4. Find names, street, and city of employees who work for 'TechM' and earn more than 10000
SELECT E.employee_name, E.street, E.city
FROM Employee E
JOIN Works W ON E.employee_name = W.employee_name
WHERE W.company_name = 'TechM' AND W.salary > 10000;

-- 5. Add Column Asset to Company table
ALTER TABLE Company ADD COLUMN Asset DECIMAL(12,2);

(6)
-- 1. Change the city of employees working with 'InfoSys' to 'Bangalore'
UPDATE Employee
SET city = 'Bangalore'
WHERE employee_name IN (
   SELECT employee_name FROM Works WHERE company_name = 'InfoSys'
```

```
);

-- 2. Find names of all employees who earn more than the average salary of their company
SELECT W1.employee_name
FROM Works W1
WHERE W1.salary > (
   SELECT AVG(W2.salary)
   FROM Works W2
   WHERE W2.company_name = W1.company_name
);

-- 3. Find names, street, and city of employees who work for 'TechM' and earn more than 10000
SELECT E.employee_name, E.street, E.city
FROM Employee E
JOIN Works W ON E.employee_name = W.employee_name
WHERE W.company_name = 'TechM' AND W.salary > 10000;

-- 4. Change the name of table Manages to Management
ALTER TABLE Manages RENAME TO Management;

-- 5. Create Simple and Unique index on Employee table
CREATE INDEX idx_employee_city ON Employee(city);           -- Simple index
CREATE UNIQUE INDEX idx_employee_name ON Employee(employee_name); -- Unique index

-- 6. Display Index Information
SHOW INDEX FROM Employee;

(7)
-- 1. Create a View1: list all customers in alphabetical order who have loan from 'Pune_Station' branch
CREATE VIEW View1 AS
SELECT DISTINCT Borrower.cust_name
FROM Borrower, Loan
WHERE Borrower.loan_no = Loan.loan_no
AND Loan.branch_name = 'Pune_Station'
ORDER BY Borrower.cust_name ASC;

-- 2. Create View2 on Branch table (any two columns)
CREATE VIEW View2 AS
SELECT branch_name, branch_city FROM Branch;

-- Perform operations on View2
INSERT INTO View2 VALUES ('Camp', 'Pune');
UPDATE View2 SET branch_city = 'Mumbai' WHERE branch_name = 'Camp';
DELETE FROM View2 WHERE branch_name = 'Camp';

-- 3. Create View3 on Borrower and Depositor selecting one column from each
CREATE VIEW View3 AS
SELECT Borrower.cust_name, Depositor.acc_no
FROM Borrower, Depositor
WHERE Borrower.cust_name = Depositor.cust_name;

-- Perform operations on View3
INSERT INTO View3 VALUES ('Aditya', 101);
UPDATE View3 SET acc_no = 102 WHERE cust_name = 'Aditya';
DELETE FROM View3 WHERE cust_name = 'Aditya';

-- 4. Create UNION of Left and Right Join for customers who have account or loan or both
SELECT c.cust_name, a.acc_no, l.loan_no
FROM Customer c
LEFT JOIN Depositor d ON c.cust_name = d.cust_name
LEFT JOIN Account a ON d.acc_no = a.acc_no
LEFT JOIN Borrower b ON c.cust_name = b.cust_name
LEFT JOIN Loan l ON b.loan_no = l.loan_no
UNION
SELECT c.cust_name, a.acc_no, l.loan_no
FROM Customer c
RIGHT JOIN Depositor d ON c.cust_name = d.cust_name
RIGHT JOIN Account a ON d.acc_no = a.acc_no
RIGHT JOIN Borrower b ON c.cust_name = b.cust_name
RIGHT JOIN Loan l ON b.loan_no = l.loan_no;

-- 5. Create Simple and Unique Index
CREATE INDEX idx_branch_city ON Branch(branch_city);          -- Simple Index
CREATE UNIQUE INDEX idx_customer_name ON Customer(cust_name); -- Unique Index

-- 6. Display Index Information
SHOW INDEX FROM Branch;
SHOW INDEX FROM Customer;

(8)
-- 1. Inner Join → shows only matching records
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
```

```
INNER JOIN Orders ON Companies.comp_id = Orders.comp_id;

-- 2. Left Outer Join → all from Companies + matching Orders
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
LEFT JOIN Orders ON Companies.comp_id = Orders.comp_id;

-- 3. Right Outer Join → all from Orders + matching Companies
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
RIGHT JOIN Orders ON Companies.comp_id = Orders.comp_id;

-- 4. Union of Left and Right Join → all records with or without match
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
LEFT JOIN Orders ON Companies.comp_id = Orders.comp_id
UNION
SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity
FROM Companies
RIGHT JOIN Orders ON Companies.comp_id = Orders.comp_id;

-- 5. Create View1 → company name and quantities
CREATE VIEW View1 AS
SELECT Companies.name, Orders.quantity
FROM Companies
JOIN Orders ON Companies.comp_id = Orders.comp_id;

-- 6. Create View2 → select any two columns from Companies table
CREATE VIEW View2 AS
SELECT comp_id, name FROM Companies;

-- Perform operations on View2
INSERT INTO View2 VALUES (5, 'Wipro');
UPDATE View2 SET name = 'Wipro Ltd' WHERE comp_id = 5;
DELETE FROM View2 WHERE comp_id = 5;

-- 7. Display contents of View1 and View2
SELECT * FROM View1;
SELECT * FROM View2;
```

(9)

```
-- 1. List all stationary items with price between 400 and 1000
SELECT * FROM ITEMS
WHERE Itype = 'Stationary' AND Iprice BETWEEN 400 AND 1000;

-- 2. Change the mobile number of customer “Gopal”
UPDATE CUSTOMERS
SET CMobile = '9998887777'
WHERE Cname = 'Gopal';

-- 3. Display the item with maximum price
SELECT Iname, Iprice
FROM ITEMS
WHERE Iprice = (SELECT MAX(Iprice) FROM ITEMS);

-- 4. Display all purchases sorted from the most recent to the oldest
SELECT * FROM PURCHASE
ORDER BY Pdate DESC;

-- 5. Count the number of customers in every city
SELECT Ccity, COUNT(*) AS Total_Customers
FROM CUSTOMERS
GROUP BY Ccity;

-- 6. Display all purchased quantity of Customer “Maya”
SELECT C.Cname, P.Pquantity, I.Iname
FROM CUSTOMERS C, PURCHASE P, ITEMS I
WHERE C.CNo = P.CNo AND P.INo = I.INo AND C.Cname = 'Maya';

-- 7. Create view which shows Iname, Price and Count of all stationary items in descending order of price
CREATE VIEW Stationary_View AS
SELECT Iname, Iprice, Icount
FROM ITEMS
WHERE Itype = 'Stationary'
ORDER BY Iprice DESC;

-- To see the view
SELECT * FROM Stationary_View;
```

(15)

```
-- Create table to store results
CREATE TABLE areas (
    radius NUMBER(5,2),
    area   NUMBER(10,2)
);

-- PL/SQL block to calculate area of circle for radius 5 to 9
DECLARE
    r NUMBER := 5;
    a NUMBER;
BEGIN
    WHILE r <= 9 LOOP
        a := 3.14 * r * r; -- area = π * r²
        INSERT INTO areas VALUES (r, a);
        r := r + 1;
    END LOOP;
    COMMIT;
END;
/
-- Display the table content
SELECT * FROM areas;
```

(16)

```
-- Create tables
CREATE TABLE Borrower1 (
  Rollin NUMBER PRIMARY KEY,
  Name VARCHAR2(50),
  DateofIssue DATE,
  NameofBook VARCHAR2(50),
  Status CHAR(1)
);

CREATE TABLE Fine1 (
  Roll_no NUMBER,
  FineDate DATE,
  Amt NUMBER
);

-- Insert sample data
INSERT INTO Borrower1 VALUES (1, 'Aditya', TO_DATE('2025-10-10','YYYY-MM-DD'), 'OS', 'I');
INSERT INTO Borrower1 VALUES (2, 'Maya', TO_DATE('2025-10-25','YYYY-MM-DD'), 'DBMS', 'I');
COMMIT;

-- PL/SQL block
SET SERVEROUTPUT ON;

DECLARE
  v_roll Borrower1.Rollin%TYPE;
  v_book Borrower1.NameofBook%TYPE;
  v_issue_date Borrower1.DateofIssue%TYPE;
  v_days NUMBER;
  v_fine NUMBER := 0;
BEGIN
  -- Accept input
  v_roll := &roll_no;
  v_book := '&book_name';

  -- Get issue date safely
  BEGIN
    SELECT DateofIssue INTO v_issue_date
    FROM Borrower1
    WHERE Rollin = v_roll AND NameofBook = v_book;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No record found for this roll number and book.');
      RETURN;
  END;

  -- Calculate days
  v_days := TRUNC(SYSDATE - v_issue_date);

  -- Fine calculation
  IF v_days BETWEEN 15 AND 30 THEN
    v_fine := v_days * 5;
  ELSIF v_days > 30 THEN
    v_fine := (30 * 5) + ((v_days - 30) * 50);
  ELSE
    v_fine := 0;
  END IF;

  -- Update status
  UPDATE Borrower1
```

```
  SET Status = 'R'
  WHERE Rollin = v_roll AND NameofBook = v_book;

  -- Insert fine if applicable
  IF v_fine > 0 THEN
    INSERT INTO Fine1 VALUES (v_roll, SYSDATE, v_fine);
  END IF;

  -- Display info
  DBMS_OUTPUT.PUT_LINE('Book returned successfully.');
  DBMS_OUTPUT.PUT_LINE('Total days: ' || v_days);
  IF v_fine > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Fine amount: Rs. ' || v_fine);
  ELSE
    DBMS_OUTPUT.PUT_LINE('No fine applicable.');
  END IF;

  COMMIT;
END;
/

(17)
-- Step 1: Create tables
CREATE TABLE O_RollCall1 (
  roll_no NUMBER PRIMARY KEY,
  name VARCHAR2(50)
);

CREATE TABLE N_RollCall1 (
  roll_no NUMBER,
  name VARCHAR2(50)
);

-- Step 2: Insert sample data into N_RollCall1
INSERT INTO N_RollCall1 VALUES (1, 'Aditya');
INSERT INTO N_RollCall1 VALUES (2, 'Karan');
INSERT INTO N_RollCall1 VALUES (3, 'Sneha');

-- Step 3: Insert sample data into O_RollCall1
INSERT INTO O_RollCall1 VALUES (2, 'Karan');
INSERT INTO O_RollCall1 VALUES (4, 'Riya');

COMMIT;

-- Step 4: PL/SQL block using cursor
DECLARE
  CURSOR c_new IS
    SELECT roll_no, name FROM N_RollCall1;

  v_count NUMBER;
BEGIN
  FOR rec IN c_new LOOP
    SELECT COUNT(*) INTO v_count
    FROM O_RollCall1
    WHERE roll_no = rec.roll_no;

    IF v_count = 0 THEN
      INSERT INTO O_RollCall1 (roll_no, name)
      VALUES (rec.roll_no, rec.name);
    END IF;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('Data merged successfully!');
END;
/

-- Step 5: View final merged data
SELECT * FROM O_RollCall1;

(18)
-- Step 1: Create the table
CREATE TABLE Student1 (
  Roll NUMBER PRIMARY KEY,
  Name VARCHAR2(50),
  Attendance NUMBER,
  Status VARCHAR2(20)
);

-- Step 2: Insert sample data
INSERT INTO Student1 VALUES (1, 'Aditya', 80, NULL);
INSERT INTO Student1 VALUES (2, 'Karan', 60, NULL);
INSERT INTO Student1 VALUES (3, 'Sneha', 75, NULL);
COMMIT;
```

```
-- Step 3: PL/SQL Block
DECLARE
  v_roll Student1.Roll%TYPE;
  v_attendance Student1.Attendance%TYPE;
BEGIN
  -- Accept roll number from user
  v_roll := &roll_no;

  -- Get attendance for the given roll number
  SELECT Attendance INTO v_attendance
  FROM Student1
  WHERE Roll = v_roll;

  -- Check attendance condition
  IF v_attendance < 75 THEN
    UPDATE Student1
    SET Status = 'Detained'
    WHERE Roll = v_roll;
    DBMS_OUTPUT.PUT_LINE('Term not granted.');
  ELSE
    UPDATE Student1
    SET Status = 'Not Detained'
    WHERE Roll = v_roll;
    DBMS_OUTPUT.PUT_LINE('Term granted.');
  END IF;

  COMMIT;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Error: No student found with that roll number.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An unexpected error occurred.');
END;
/

-- Step 4: To verify
SELECT * FROM Student1;
```

(19)

```
-- Step 1: Create tables
CREATE TABLE emp1 (
  emp_no NUMBER PRIMARY KEY,
  salary NUMBER
);

CREATE TABLE increment_salary1 (
  emp_no NUMBER,
  salary NUMBER
);

-- Step 2: Insert sample data
INSERT INTO emp1 VALUES (1, 25000);
INSERT INTO emp1 VALUES (2, 40000);
INSERT INTO emp1 VALUES (3, 30000);
INSERT INTO emp1 VALUES (4, 50000);
COMMIT;

-- Step 3: PL/SQL Block
DECLARE
  v_avg_salary NUMBER;
BEGIN
  -- Find average salary of all employees
  SELECT AVG(salary) INTO v_avg_salary FROM emp1;

  -- Update salaries and record changes
  FOR rec IN (SELECT emp_no, salary FROM emp1 WHERE salary < v_avg_salary) LOOP
    UPDATE emp1
    SET salary = salary + (salary * 0.10)
    WHERE emp_no = rec.emp_no;

    INSERT INTO increment_salary1 VALUES (rec.emp_no, rec.salary + (rec.salary * 0.10));
  END LOOP;

  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Salary updated and increment records inserted successfully.');
END;
/

-- Step 4: To verify
SELECT * FROM emp1;
SELECT * FROM increment_salary1;
```

(20)

```
-- Step 1: Create tables
CREATE TABLE Stud_Marks1 (
  Roll NUMBER PRIMARY KEY,
  Name VARCHAR2(50),
  Total_Marks NUMBER
);

CREATE TABLE Result1 (
  Roll NUMBER,
  Name VARCHAR2(50),
  Class VARCHAR2(30)
);

-- Step 2: Insert sample data
INSERT INTO Stud_Marks1 VALUES (1, 'Aditya', 1200);
INSERT INTO Stud_Marks1 VALUES (2, 'Rohan', 950);
INSERT INTO Stud_Marks1 VALUES (3, 'Sahil', 870);
INSERT INTO Stud_Marks1 VALUES (4, 'Karan', 800);
COMMIT;

-- Step 3: Create Stored Procedure
CREATE OR REPLACE PROCEDURE proc_Grade IS
  v_class VARCHAR2(30);
BEGIN
  FOR rec IN (SELECT * FROM Stud_Marks1) 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';
    ELSE
      v_class := 'Fail';
    END IF;

    INSERT INTO Result1 VALUES (rec.Roll, rec.Name, v_class);
  END LOOP;
  COMMIT;
END;
/

-- Step 4: Execute procedure using PL/SQL block
BEGIN
  proc_Grade;
  DBMS_OUTPUT.PUT_LINE('Grades calculated and inserted into Result1 successfully.');
END;
/

-- Step 5: View results
SELECT * FROM Result1;
```

(21)

```
-- Step 1: Create the function
CREATE OR REPLACE FUNCTION Age_calc(
  p_dob DATE,               -- Input parameter: Date of Birth
  p_months OUT NUMBER,      -- OUT parameter: Months
  p_days OUT NUMBER         -- OUT parameter: Days
) RETURN NUMBER             -- Return type: Years
IS
  v_years NUMBER;
  v_total_months NUMBER;
  v_temp_date DATE := SYSDATE;
BEGIN
  -- Calculate total months between DOB and current date
  v_total_months := MONTHS_BETWEEN(v_temp_date, p_dob);

  -- Separate into years and remaining months
  v_years := FLOOR(v_total_months / 12);
  p_months := FLOOR(MOD(v_total_months, 12));

  -- Find days difference (remaining days after complete months)
  p_days := TRUNC(v_temp_date - ADD_MONTHS(p_dob, (v_years * 12 + p_months)));

  RETURN v_years;
END;
/

-- Step 2: Test the function
SET SERVEROUTPUT ON;
DECLARE
  v_years NUMBER;
  v_months NUMBER;
```

```
  v_days NUMBER;
BEGIN
  v_years := Age_calc(TO_DATE('15-08-2000', 'DD-MM-YYYY'), v_months, v_days);
  DBMS_OUTPUT.PUT_LINE('Age: ' || v_years || ' years, ' || v_months || ' months, ' || v_days || ' days');
END;
/
```

(22)

```
-- Step 1: Create main table
CREATE TABLE Library (
  Book_ID NUMBER PRIMARY KEY,
  Book_Name VARCHAR2(100)
);

-- Step 2: Create audit table
CREATE TABLE Library_Audit (
  Audit_ID NUMBER GENERATED ALWAYS AS IDENTITY,
  Book_ID NUMBER,
  Book_Name VARCHAR2(100),
  Action_Type VARCHAR2(20),
  Action_Date DATE
);

-- Step 3: BEFORE trigger
CREATE OR REPLACE TRIGGER trg_library_before
BEFORE UPDATE OR DELETE ON Library
FOR EACH ROW
BEGIN
  INSERT INTO Library_Audit (Book_ID, Book_Name, Action_Type, Action_Date)
  VALUES (:OLD.Book_ID, :OLD.Book_Name,
          CASE
            WHEN DELETING THEN 'BEFORE DELETE'
            WHEN UPDATING THEN 'BEFORE UPDATE'
          END,
          SYSDATE);
END;
/

-- Step 4: AFTER trigger
CREATE OR REPLACE TRIGGER trg_library_after
AFTER UPDATE OR DELETE ON Library
FOR EACH ROW
BEGIN
  INSERT INTO Library_Audit (Book_ID, Book_Name, Action_Type, Action_Date)
  VALUES (:OLD.Book_ID, :OLD.Book_Name,
          CASE
            WHEN DELETING THEN 'AFTER DELETE'
            WHEN UPDATING THEN 'AFTER UPDATE'
          END,
          SYSDATE);
END;
/
```

(23)

```
-- Step 1: Create the CUSTOMERS table
CREATE TABLE CUSTOMERS (
  C_ID NUMBER PRIMARY KEY,
  C_NAME VARCHAR2(50),
  SALARY NUMBER
);

-- Step 2: Create the Trigger
CREATE OR REPLACE TRIGGER trg_salary_diff
AFTER INSERT OR UPDATE OR DELETE ON CUSTOMERS
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('New record inserted. Salary: ' || :NEW.SALARY);

  ELSIF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('Salary updated. Old Salary: ' || :OLD.SALARY ||
                         ', New Salary: ' || :NEW.SALARY ||
                         ', Difference: ' || (:NEW.SALARY - :OLD.SALARY));

  ELSIF DELETING THEN
    DBMS_OUTPUT.PUT_LINE('Record deleted. Previous Salary: ' || :OLD.SALARY);
  END IF;
END;
/
```

(24)

```
-- Main Employee table
CREATE TABLE Emp (
```

```
  Emp_no NUMBER PRIMARY KEY,
  Emp_name VARCHAR2(50),
  Emp_salary NUMBER
);

-- Tracking table to store salaries less than 50,000
CREATE TABLE Tracking (
  Emp_no NUMBER,
  Emp_salary NUMBER
);

CREATE OR REPLACE TRIGGER trg_salary_tracking
AFTER INSERT OR UPDATE ON Emp
FOR EACH ROW
BEGIN
  -- For INSERT: check if salary < 50,000
  IF INSERTING AND :NEW.Emp_salary < 50000 THEN
    DBMS_OUTPUT.PUT_LINE('Warning: Inserted salary less than 50,000: ' || :NEW.Emp_salary);
    INSERT INTO Tracking(Emp_no, Emp_salary) VALUES (:NEW.Emp_no, :NEW.Emp_salary);
  END IF;

  -- For UPDATE: check if new salary < 50,000
  IF UPDATING AND :NEW.Emp_salary < 50000 THEN
    DBMS_OUTPUT.PUT_LINE('Warning: Updated salary less than 50,000: ' || :NEW.Emp_salary);
    INSERT INTO Tracking(Emp_no, Emp_salary) VALUES (:NEW.Emp_no, :NEW.Emp_salary);
  END IF;
END;
/
```

(25)

```
from pymongo import MongoClient

# Connect to MongoDB (local server)
client = MongoClient("mongodb://localhost:27017/")
db = client['mydatabase']         # Database
collection = db['employees']      # Collection

def add_employee():
    emp_no = int(input("Enter Employee No: "))
    name = input("Enter Employee Name: ")
    salary = float(input("Enter Salary: "))
    record = {"Emp_no": emp_no, "Emp_name": name, "Emp_salary": salary}
    collection.insert_one(record)
    print("Employee added successfully!")

def view_employees():
    print("All Employees:")
    for emp in collection.find():
        print(emp)

def update_employee():
    emp_no = int(input("Enter Employee No to update: "))
    new_salary = float(input("Enter new salary: "))
    result = collection.update_one({"Emp_no": emp_no}, {"$set": {"Emp_salary": new_salary}})
    if result.matched_count:
        print("Employee salary updated successfully!")
    else:
        print("Employee not found.")

def delete_employee():
    emp_no = int(input("Enter Employee No to delete: "))
    result = collection.delete_one({"Emp_no": emp_no})
    if result.deleted_count:
        print("Employee deleted successfully!")
    else:
        print("Employee not found.")

# Menu-driven interface
while True:
    print("\n--- Employee Management ---")
    print("1. Add Employee")
    print("2. View Employees")
    print("3. Update Employee")
    print("4. Delete Employee")
    print("5. Exit")

    choice = input("Enter your choice: ")

    if choice == '1':
        add_employee()
    elif choice == '2':
        view_employees()
    elif choice == '3':
```

```
        update_employee()
    elif choice == '4':
        delete_employee()
    elif choice == '5':
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Try again.")
```

(26)

```
import mysql.connector

# Connect to MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password",
    database="company"
)
cursor = conn.cursor()
cursor.execute("""
CREATE TABLE IF NOT EXISTS employees (
    emp_no INT PRIMARY KEY,
    emp_name VARCHAR(50),
    emp_salary FLOAT
)
""")

def add_employee():
    cursor.execute("INSERT INTO employees VALUES (%s,%s,%s)",
                   (int(input("Emp No: ")), input("Name: "), float(input("Salary: "))))
    conn.commit()
    print("Added successfully!")

def view_employees():
    cursor.execute("SELECT * FROM employees")
    for row in cursor.fetchall():
        print(row)

def update_employee():
    cursor.execute("UPDATE employees SET emp_salary=%s WHERE emp_no=%s",
                   (float(input("New Salary: ")), int(input("Emp No to update: "))))
    conn.commit()
    print("Updated successfully!" if cursor.rowcount else "Employee not found.")

def delete_employee():
    cursor.execute("DELETE FROM employees WHERE emp_no=%s", (int(inpupt("Emp No to delete: ")),))
    conn.commit()
    print("Deleted successfully!" if cursor.rowcount else "Employee not found.")

while True:
    print("\n1.Add 2.View 3.Update 4.Delete 5.Exit")
    choice = input("Choice: ")
    if choice=='1': add_employee()
    elif choice=='2': view_employees()
    elif choice=='3': update_employee()
    elif choice=='4': delete_employee()
    elif choice=='5': break
    else: print("Invalid choice.")

cursor.close()
conn.close()
```

(10)

```
// 1. Create Employee Collection and Insert Documents
db.Employee.insertMany([
  {
    Name: { FName: "Aditya", LName: "Wagh" },
    Company_Name: "TCS",
    Salary: 45000,
    Designation: "Programmer",
    Age: 24,
    Expertise: ["MongoDB", "NodeJS", "Express"],
    DOB: "2001-03-15",
    Email: "aditya.wagh@example.com",
    Contact: "9876543210",
    Address: [{ PAddr: "Pune", LAddr: "Mumbai" }]
  },
  {
    Name: { FName: "Shreyas", LName: "Patil" },
    Company_Name: "Infosys",
    Salary: 52000,
    Designation: "Developer",
```

```
    Age: 26,
    Expertise: ["Python", "Flask", "MySQL"],
    DOB: "1999-08-20",
    Email: "shreyas.patil@example.com",
    Contact: "9123456780",
    Address: [{ PAddr: "Nagpur", LAddr: "Pune" }]
   },
   {
    Name: { FName: "Samiksha", LName: "Deshmukh" },
    Company_Name: "TechM",
    Salary: 38000,
    Designation: "Programmer",
    Age: 23,
    Expertise: ["JavaScript", "React", "MongoDB"],
    DOB: "2002-01-25",
    Email: "samiksha.deshmukh@example.com",
    Contact: "9988776655",
    Address: [{ PAddr: "Nashik", LAddr: "Mumbai" }]
   },
   {
    Name: { FName: "Harsh", LName: "Shinde" },
    Company_Name: "TCS",
    Salary: 42000,
    Designation: "Tester",
    Age: 25,
    Expertise: ["Selenium", "Java", "MySQL"],
    DOB: "2000-12-10",
    Email: "harsh.shinde@example.com",
    Contact: "9090909090",
    Address: [{ PAddr: "Pune", LAddr: "Pune" }]
   },
   {
    Name: { FName: "Mansi", LName: "Kulkarni" },
    Company_Name: "Infosys",
    Salary: 60000,
    Designation: "Manager",
    Age: 28,
    Expertise: ["MongoDB", "MySQL", "Cassandra"],
    DOB: "1997-04-05",
    Email: "mansi.kulkarni@example.com",
    Contact: "9876501234",
    Address: [{ PAddr: "Mumbai", LAddr: "Mumbai" }]
   }
]);

// 2. Query: Select all documents where Designation = "Programmer" and Salary > 30000
db.Employee.find({ Designation: "Programmer", Salary: { $gt: 30000 } });

// 3. Query: Create new document if no document has given fields (upsert)
db.Employee.updateOne(
  { Designation: "Tester", Company_Name: "TCS", Age: 25 },
  {
    $setOnInsert: {
      Name: { FName: "Gopal", LName: "Rao" },
      Salary: 35000,
      Expertise: ["Manual Testing", "Automation"],
      DOB: "2000-09-15",
      Email: "gopal.rao@example.com",
      Contact: "9812345678",
      Address: [{ PAddr: "Hyderabad", LAddr: "Hyderabad" }]
    }
  },
  { upsert: true }
);

// 4. Query: Increase salary of each Employee working with "Infosys" by 10000
db.Employee.updateMany({ Company_Name: "Infosys" }, { $inc: { Salary: 10000 } });

// 5. Query: Reduce salary by 5000 for all employees working with "TCS"
db.Employee.updateMany({ Company_Name: "TCS" }, { $inc: { Salary: -5000 } });

// 6. Query: Return documents where Designation != "Tester"
db.Employee.find({ Designation: { $ne: "Tester" } });

// 7. Query: Find all employees with exact array match on Expertise
db.Employee.find({ Expertise: ["MongoDB", "MySQL", "Cassandra"] });
```

(11)

```
// 1. Final name of Employee where age is less than 30 and salary more than 50000
db.Employee.find(
  { age: { $lt: 30 }, salary: { $gt: 50000 } },
  { "Name.FName": 1, "Name.LName": 1, _id: 0 }
)
```

```
// 2. Create a new document if not exists
db.Employee.updateOne(
  { Designation: "Tester", Company_name: "TCS", Age: 25 },
  {
    $setOnInsert: {
      Name: { FName: "New", LName: "Tester" },
      Salary: 35000,
      Expertise: ["Manual Testing", "Automation"],
      DOB: "2000-05-10",
      Email: "tester@tcs.com",
      Contact: "9999999999",
      Address: [{ PAddr: "Pune", LAddr: "Mumbai" }]
    }
  },
  { upsert: true }
)

// 3. Select all documents where age < 30 or salary > 40000
db.Employee.find(
  { $or: [{ age: { $lt: 30 } }, { salary: { $gt: 40000 } }] }
)

// 4. Find documents where Designation is not equal to "Developer"
db.Employee.find(
  { Designation: { $ne: "Developer" } }
)

// 5. Find _id, Designation, Address, and Name where Company_name = "Infosys"
db.Employee.find(
  { Company_name: "Infosys" },
  { _id: 1, Designation: 1, Address: 1, Name: 1 }
)

// 6. Display only FName and LName of all Employees
db.Employee.find(
  {},
  { "Name.FName": 1, "Name.LName": 1, _id: 0 }
)

// 1. Create a new document if not exists
db.Employee.updateOne(
  { Designation: "Tester", Company_name: "TCS", Age: 25 },
  {
    $setOnInsert: {
      Emp_id: 101,
      Name: { FName: "Test", LName: "User" },
      Salary: 30000,
      Expertise: ["Manual Testing", "Automation"],
      DOB: "2000-01-01",
      Email: "test.user@tcs.com",
      Contact: "9876543210",
      Address: [{ PAddr: "Pune", LAddr: "Mumbai" }]
    }
  },
  { upsert: true }
)

// 2. Increase salary by 2000 for all employees in TCS
db.Employee.updateMany(
  { Company_name: "TCS" },
  { $inc: { Salary: 2000 } }
)

// 3. Match documents with Address having { city: "Pune", Pin_code: "411001" }
db.Employee.find({
  Address: { $elemMatch: { city: "Pune", Pin_code: "411001" } }
})

// 4. Find employee details who are Developer or Tester
db.Employee.find({
  Designation: { $in: ["Developer", "Tester"] }
})

// 5. Drop (delete) a single document where Designation = "Developer"
db.Employee.deleteOne({ Designation: "Developer" })

// 6. Count total number of documents in Employee collection
db.Employee.countDocuments()
```

(12)

```
// Return Designation with Total Salary above 200000
db.Employee.aggregate([
```

```
  { $group: { _id: "$Designation", TotalSalary: { $sum: "$Salary" } } },
  { $match: { TotalSalary: { $gt: 200000 } } }
])

//Return names and _id in uppercase and alphabetical order
db.Employee.aggregate([
  {
   $project: {
     _id: { $toUpper: { $toString: "$_id" } },
     FName: { $toUpper: "$Name.FName" },
     LName: { $toUpper: "$Name.LName" }
   }
  },
  { $sort: { FName: 1 } }
])

//Find total salary for each city with Designation = "DBA"
db.Employee.aggregate([
  { $match: { Designation: "DBA" } },
  { $unwind: "$Address" },
  { $group: { _id: "$Address.PAddr", TotalSalary: { $sum: "$Salary" } } }
])

//Create Single Field Index on Designation
db.Employee.createIndex({ Designation: 1 })

//Create Multikey Index on Expertise field
db.Employee.createIndex({ Expertise: 1 })

//Create Index on Emp_id and compare time before and after
// Before index
db.Employee.find({ Emp_id: 5000 }).explain("executionStats")

//Create index
db.Employee.createIndex({ Emp_id: 1 })

//After index
db.Employee.find({ Emp_id: 5000 }).explain("executionStats")

// Return list of indexes created on Employee collection
db.Employee.getIndexes()
```

(13)

```
//Using aggregation – separate values in Expertise array and return sum of each element
db.Employee.aggregate([
  { $unwind: "$Expertise" },
  { $group: { _id: "$Expertise", Total: { $sum: 1 } } }
])

//Using aggregation – return Max and Min Salary for each company
db.Employee.aggregate([
  {
   $group: {
     _id: "$Company_Name",
     MaxSalary: { $max: "$Salary" },
     MinSalary: { $min: "$Salary" }
   }
  }
])

//Using aggregation – total salary for each city with Designation = "DBA"
db.Employee.aggregate([
  { $match: { Designation: "DBA" } },
  { $unwind: "$Address" },
  { $group: { _id: "$Address.PAddr", TotalSalary: { $sum: "$Salary" } } }
])

//Using aggregation – separate values in Expertise array for employee named "Swapnil Jadhav"
db.Employee.aggregate([
  { $match: { "Name.FName": "Swapnil", "Name.LName": "Jadhav" } },
  { $unwind: "$Expertise" },
  { $project: { _id: 0, "Expertise": 1 } }
])

//Create Compound Index on Name and Age
db.Employee.createIndex({ "Name.FName": 1, Age: -1 })

//Create Index on Emp_id and compare search time before and after index
//Before index
db.Employee.find({ Emp_id: 5000 }).explain("executionStats")

// Create index
db.Employee.createIndex({ Emp_id: 1 })
```

```
// After index
db.Employee.find({ Emp_id: 5000 }).explain("executionStats")

//Return list of indexes created on Employee collection
db.Employee.getIndexes()
```

(14)

```
// Total salary per company
db.Employee.mapReduce(
  function() { emit(this["Company Name"], this.Salary); },
  function(key, values) { return Array.sum(values); },
  { out: "TotalSalaryPerCompany" }
)

//Total salary for company "TCS"
db.Employee.mapReduce(
  function() { if (this["Company Name"] === "TCS") emit(this["Company Name"], this.Salary); },
  function(key, values) { return Array.sum(values); },
  { out: "TotalSalaryTCS" }
)

//Average salary of company whose address is “Pune”
db.Employee.mapReduce(
  function() {
    this.Address.forEach(a => {
      if (a.PAddr === "Pune") emit(this["Company Name"], this.Salary);
    });
  },
  function(key, values) { return Array.avg(values); },
  { out: "AvgSalaryPune" }
)

//Total count for “City = Pune”
db.Employee.mapReduce(
  function() {
    this.Address.forEach(a => {
      if (a.PAddr === "Pune") emit("PuneCount", 1);
    });
  },
  function(key, values) { return Array.sum(values); },
  { out: "CityPuneCount" }
)

//Count for city Pune and age greater than 40
db.Employee.mapReduce(
  function() {
    this.Address.forEach(a => {
      if (a.PAddr === "Pune" && this.Age > 40) emit("Pune_Age>40", 1);
    });
  },
  function(key, values) { return Array.sum(values); },
  { out: "PuneAgeAbove40Count" }
)
```