|  |  |
| --- | --- |
| **S1** | Consider following Relation  **Account (Acc\_no, branch\_name,balance)**  **Branch(branch\_name,branch\_city,assets)**  **Customer(cust\_name,cust\_street,cust\_city)**  **Depositor(cust\_name,acc\_no)**  **Loan(loan\_no,branch\_name,amount)**  **Borrower(cust\_name,loan\_no)**  -- Account table  CREATE TABLE Account (  Acc\_no INT PRIMARY KEY,  branch\_name VARCHAR(50) NOT NULL,  balance DECIMAL(10, 2) NOT NULL,  FOREIGN KEY (branch\_name) REFERENCES Branch(branch\_name)  );  -- Branch table  CREATE TABLE Branch (  branch\_name VARCHAR(50) PRIMARY KEY,  branch\_city VARCHAR(50) NOT NULL,  assets DECIMAL(15, 2) NOT NULL  );  -- Customer table  CREATE TABLE Customer (  cust\_name VARCHAR(50) PRIMARY KEY,  cust\_street VARCHAR(50) NOT NULL,  cust\_city VARCHAR(50) NOT NULL  );  -- Depositor table (association between Customer and Account)  CREATE TABLE Depositor (  cust\_name VARCHAR(50),  acc\_no INT,  PRIMARY KEY (cust\_name, acc\_no),  FOREIGN KEY (cust\_name) REFERENCES Customer(cust\_name),  FOREIGN KEY (acc\_no) REFERENCES Account(Acc\_no)  );  -- Loan table  CREATE TABLE Loan (  loan\_no INT PRIMARY KEY,  branch\_name VARCHAR(50) NOT NULL,  amount DECIMAL(10, 2) NOT NULL,  FOREIGN KEY (branch\_name) REFERENCES Branch(branch\_name)  );  -- Borrower table (association between Customer and Loan)  CREATE TABLE Borrower (  cust\_name VARCHAR(50),  loan\_no INT,  PRIMARY KEY (cust\_name, loan\_no),  FOREIGN KEY (cust\_name) REFERENCES Customer(cust\_name),  FOREIGN KEY (loan\_no) REFERENCES Loan(loan\_no)  );  Create above tables with appropriate constraints like primary key, foreign key, not null etc.   1. Find the names of all branches in loan relation.   SELECT DISTINCT branch\_name FROM Loan;   1. Find all loan numbers for loans made at ‘Wadia College’ Branch with loan amount > 12000.   SELECT loan\_no  FROM Loan  WHERE branch\_name = 'Wadia College' AND amount > 12000;   1. Find all customers who have a loan from bank. Find their names,loan\_no and loan amount.   SELECT Customer.cust\_name, Loan.loan\_no, Loan.amount  FROM Customer  JOIN Borrower ON Customer.cust\_name = Borrower.cust\_name  JOIN Loan ON Borrower.loan\_no = Loan.loan\_no;   1. List all customers in alphabetical order who have loan from ‘Wadia College’ branch.   SELECT Customer.cust\_name  FROM Customer  JOIN Borrower ON Customer.cust\_name = Borrower.cust\_name  JOIN Loan ON Borrower.loan\_no = Loan.loan\_no  WHERE Loan.branch\_name = 'Wadia College'  ORDER BY Customer.cust\_name;   1. Display distinct cities of branch.   SELECT DISTINCT branch\_city FROM Branch; |
|  |  |
| **S2** | Consider following Relation  **Account (Acc\_no, branch\_name,balance)**  **Branch(branch\_name,branch\_city,assets)**  **Customer(cust\_name,cust\_street,cust\_city)**  **Depositor(cust\_name,acc\_no)**  **Loan(loan\_no,branch\_name,amount)**  **Borrower(cust\_name,loan\_no)**  Create above tables with appropriate constraints like primary key, foreign key, not null etc.   1. Find all customers who have both account and loan at bank.   SELECT DISTINCT Depositor.cust\_name  FROM Depositor  JOIN Borrower ON Depositor.cust\_name = Borrower.cust\_name;   1. Find all customers who have an account or loan or both at bank.   SELECT DISTINCT cust\_name  FROM (  SELECT cust\_name FROM Depositor  UNION  SELECT cust\_name FROM Borrower  ) AS AllCustomers;   1. Find all customers who have account but no loan at the bank.   SELECT DISTINCT Depositor.cust\_name  FROM Depositor  LEFT JOIN Borrower ON Depositor.cust\_name = Borrower.cust\_name  WHERE Borrower.cust\_name IS NULL;   1. Find average account balance at ‘Wadia College’ branch.   SELECT AVG(balance) AS average\_balance  FROM Account  WHERE branch\_name = 'Wadia College';   1. Find no. of depositors at each branch   SELECT Account.branch\_name, COUNT(DISTINCT Depositor.cust\_name) AS num\_depositors  FROM Account  JOIN Depositor ON Account.Acc\_no = Depositor.acc\_no  GROUP BY Account.branch\_name; |
|  |  |
| **P10** | **Trigger:** Write a after trigger for Insert, update and delete event considering following requirement:  Emp(Emp\_no, Emp\_name, Emp\_salary)  -- Emp table  CREATE TABLE Emp (  Emp\_no INT PRIMARY KEY,  Emp\_name VARCHAR(50) NOT NULL,  Emp\_salary DECIMAL(10, 2) NOT NULL  );  -- Tracking table  CREATE TABLE Tracking (  Emp\_no INT,  Emp\_salary DECIMAL(10, 2),  FOREIGN KEY (Emp\_no) REFERENCES Emp(Emp\_no)  );   1. Trigger should be initiated when salary tried to be inserted is less than Rs.50,000/-   -- Trigger for INSERT and UPDATE operations  CREATE TRIGGER trg\_track\_low\_salary  AFTER INSERT OR UPDATE ON Emp  FOR EACH ROW  BEGIN  -- Check if the inserted or updated salary is less than 50,000  IF NEW.Emp\_salary < 50000 THEN  -- Insert the employee details into the Tracking table  INSERT INTO Tracking (Emp\_no, Emp\_salary)  VALUES (NEW.Emp\_no, NEW.Emp\_salary);  END IF;  END;   1. Trigger should be initiated when salary tried to be updated for value less than Rs. 50,000/-   -- Trigger for DELETE operation  CREATE TRIGGER trg\_track\_deleted\_salary  AFTER DELETE ON Emp  FOR EACH ROW  BEGIN  -- Log the deleted record (optional)  INSERT INTO Tracking (Emp\_no, Emp\_salary)  VALUES (OLD.Emp\_no, OLD.Emp\_salary);  END;  Also the new values expected to be inserted will be stored in new table Tracking(Emp\_no,Emp\_salary). |
|  |  |

|  |  |
| --- | --- |
| **S3** | Consider following Relation  **Account (Acc\_no, branch\_name,balance)**  **Branch(branch\_name,branch\_city,assets)**  **Customer(cust\_name,cust\_street,cust\_city)**  **Depositor(cust\_name,acc\_no)**  **Loan(loan\_no,branch\_name,amount)**  **Borrower(cust\_name,loan\_no)**  Create above tables with appropriate constraints like primary key, foreign key, not null etc.   1. Find the branches where average account balance > 15000.   SELECT branch\_name  FROM Account  GROUP BY branch\_name  HAVING AVG(balance) > 15000;   1. Find number of tuples in customer relation.   SELECT COUNT(\*) AS num\_of\_customers  FROM Customer;   1. Calculate total loan amount given by bank.   SELECT SUM(amount) AS total\_loan\_amount  FROM Loan;   1. Delete all loans with loan amount between 1300 and 1500.   DELETE FROM Loan  WHERE amount BETWEEN 1300 AND 1500;   1. Find the average account balance at each branch   SELECT branch\_name, AVG(balance) AS average\_balance  FROM Account  GROUP BY branch\_name;   1. Find name of Customer and city where customer name starts with Letter P.   SELECT cust\_name, cust\_city  FROM Customer  WHERE cust\_name LIKE 'P%'; |
|  |  |
| **S4** | **SQL Queries:**  Create following tables with suitable constraints (primary key, foreign key, not null etc).  Insert record and solve the following queries:  **Create table Cust\_Master(Cust\_no, Cust\_name, Cust\_addr)**  **Create table Order(Order\_no, Cust\_no, Order\_date, Qty\_Ordered)**  **Create Product (Product\_no, Product\_name, Order\_no)**  **-- Cust\_Master table**  **CREATE TABLE Cust\_Master (**  **Cust\_no VARCHAR(10) PRIMARY KEY,**  **Cust\_name VARCHAR(50) NOT NULL,**  **Cust\_addr VARCHAR(100) NOT NULL**  **);**  **-- Order table**  **CREATE TABLE Orders (**  **Order\_no INT PRIMARY KEY,**  **Cust\_no VARCHAR(10),**  **Order\_date DATE NOT NULL,**  **Qty\_Ordered INT NOT NULL,**  **FOREIGN KEY (Cust\_no) REFERENCES Cust\_Master(Cust\_no)**  **);**  **-- Product table**  **CREATE TABLE Product (**  **Product\_no INT PRIMARY KEY,**  **Product\_name VARCHAR(50) NOT NULL,**  **Order\_no INT,**  **FOREIGN KEY (Order\_no) REFERENCES Orders(Order\_no)**  **);**  **-- Sample Data Insertion**  **INSERT INTO Cust\_Master (Cust\_no, Cust\_name, Cust\_addr) VALUES**  **('C1001', 'Adam', 'Banglore'),**  **('C1002', 'Sara', 'Mumbai'),**  **('C1003', 'Ravi', 'Manglore'),**  **('C1004', 'Amanda', 'Delhi'),**  **('C1005', 'Raj', 'Banglore');**  **INSERT INTO Orders (Order\_no, Cust\_no, Order\_date, Qty\_Ordered) VALUES**  **(1, 'C1001', '2023-11-01', 5),**  **(2, 'C1002', '2023-11-02', 10),**  **(3, 'C1003', '2023-11-03', 8),**  **(4, 'C1004', '2023-11-04', 3),**  **(5, 'C1005', '2023-11-05', 7);**  **INSERT INTO Product (Product\_no, Product\_name, Order\_no) VALUES**  **(101, 'Laptop', 1),**  **(102, 'Tablet', 2),**  **(103, 'Monitor', 3),**  **(104, 'Mouse', 4),**  **(105, 'Keyboard', 5);**   1. List names of customers having 'A' as second letter in their name.   SELECT Cust\_name  FROM Cust\_Master  WHERE Cust\_name LIKE '\_A%';   1. Display order from Customer no C1002, C1005, C1007 and C1008   SELECT \*  FROM Orders  WHERE Cust\_no IN ('C1002', 'C1005', 'C1007', 'C1008');   1. List Clients who stay in either 'Banglore or 'Manglore'   SELECT Cust\_name  FROM Cust\_Master  WHERE Cust\_addr IN ('Banglore', 'Manglore');   1. Display name of customers& the product\_name they have purchase   SELECT CM.Cust\_name, P.Product\_name  FROM Cust\_Master AS CM  JOIN Orders AS O ON CM.Cust\_no = O.Cust\_no  JOIN Product AS P ON O.Order\_no = P.Order\_no;   1. Create view View1 consisting of Cust\_name, Product\_name.   CREATE VIEW View1 AS  SELECT CM.Cust\_name, P.Product\_name  FROM Cust\_Master AS CM  JOIN Orders AS O ON CM.Cust\_no = O.Cust\_no  JOIN Product AS P ON O.Order\_no = P.Order\_no;   1. Disply product\_name and quantity purchase by each customer   SELECT CM.Cust\_name, P.Product\_name, O.Qty\_Ordered  FROM Cust\_Master AS CM  JOIN Orders AS O ON CM.Cust\_no = O.Cust\_no  JOIN Product AS P ON O.Order\_no = P.Order\_no;   1. Perform different joint operation.   Inner Join: List customers with their orders.  SELECT CM.Cust\_name, O.Order\_no, O.Order\_date, O.Qty\_Ordered  FROM Cust\_Master AS CM  JOIN Orders AS O ON CM.Cust\_no = O.Cust\_no;  Left Outer Join: List all customers with their orders, if any.  SELECT CM.Cust\_name, O.Order\_no, O.Order\_date, O.Qty\_Ordered  FROM Cust\_Master AS CM  LEFT JOIN Orders AS O ON CM.Cust\_no = O.Cust\_no;  Right Outer Join: List all orders with customer information, if available.  SELECT CM.Cust\_name, O.Order\_no, O.Order\_date, O.Qty\_Ordered  FROM Cust\_Master AS CM  RIGHT JOIN Orders AS O ON CM.Cust\_no = O.Cust\_no;  Full Outer Join (if supported by the DBMS): List all customers and all orders, matching where possible.  SELECT CM.Cust\_name, O.Order\_no, O.Order\_date, O.Qty\_Ordered  FROM Cust\_Master AS CM  FULL OUTER JOIN Orders AS O ON CM.Cust\_no = O.Cust\_no; |
|  |  |
| **P1** | Write a **PL/SQL code** block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area.  CREATE TABLE areas (  radius DECIMAL(5, 2),  area DECIMAL(10, 2)  );  DECLARE  radius DECIMAL(5, 2);  area DECIMAL(10, 2);  BEGIN  -- Loop through radius values from 5 to 9  FOR radius IN 5..9 LOOP  -- Calculate the area for the given radius (Area = π \* r^2)  area := 3.14159 \* radius \* radius;    -- Insert the radius and calculated area into the areas table  INSERT INTO areas (radius, area)  VALUES (radius, area);  END LOOP;    -- Commit the changes  COMMIT;    -- Output a success message  DBMS\_OUTPUT.PUT\_LINE('Areas calculated and stored successfully.');  END;  / |
|  |  |
| **P3** | Write a **PL/SQL block** of code using Cursor that will merge the data available in the newly created table N\_Roll Call 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.  DECLARE  -- Declare cursor to select data from N\_RollCall table  CURSOR cur\_rollcall IS  SELECT roll\_no, student\_name  FROM N\_RollCall;  -- Declare variables to hold cursor data  v\_roll\_no INT;  v\_student\_name VARCHAR(50);  BEGIN  -- Open the cursor  OPEN cur\_rollcall;  -- Loop through each record in the cursor  LOOP  FETCH cur\_rollcall INTO v\_roll\_no, v\_student\_name;  EXIT WHEN cur\_rollcall%NOTFOUND;  -- Check if the record already exists in O\_RollCall table  BEGIN  -- If the record doesn't exist, insert it into O\_RollCall  IF NOT EXISTS (SELECT 1 FROM O\_RollCall WHERE roll\_no = v\_roll\_no) THEN  INSERT INTO O\_RollCall (roll\_no, student\_name)  VALUES (v\_roll\_no, v\_student\_name);  END IF;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  -- Skip insertion if duplicate value error occurs (e.g., unique constraint violation)  NULL;  END;  END LOOP;  -- Close the cursor  CLOSE cur\_rollcall;  -- Commit the changes  COMMIT;  -- Output a success message  DBMS\_OUTPUT.PUT\_LINE('Data from N\_RollCall merged into O\_RollCall successfully.');  END;  / |

|  |  |
| --- | --- |
| **S5** | Consider following Relation  **Employee(emp\_id,employee\_name,street,city)**  **Works(employee\_name,company\_name,salary)**  **Company(company\_name,city)**  **Manages(employee\_name,manager\_name)**  -- Employee table  CREATE TABLE Employee (  emp\_id INT PRIMARY KEY,  employee\_name VARCHAR(50) NOT NULL UNIQUE,  street VARCHAR(50),  city VARCHAR(50)  );  -- Company table  CREATE TABLE Company (  company\_name VARCHAR(50) PRIMARY KEY,  city VARCHAR(50) NOT NULL  );  -- Works table  CREATE TABLE Works (  employee\_name VARCHAR(50) NOT NULL,  company\_name VARCHAR(50) NOT NULL,  salary DECIMAL(10, 2) NOT NULL,  FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name),  FOREIGN KEY (company\_name) REFERENCES Company(company\_name)  );  -- Manages table  CREATE TABLE Manages (  employee\_name VARCHAR(50) NOT NULL,  manager\_name VARCHAR(50),  FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name),  FOREIGN KEY (manager\_name) REFERENCES Employee(employee\_name)  );  Create above tables with appropriate constraints like primary key, foreign key, not null etc.   1. Find the names of all employees who work for ‘TCS’.   SELECT employee\_name  FROM Works  WHERE company\_name = 'TCS';   1. Find the names and company names of all employees sorted in ascending order of company name and descending order of employee names of that company.   SELECT employee\_name, company\_name  FROM Works  ORDER BY company\_name ASC, employee\_name DESC;   1. Change the city of employee working with InfoSys to ‘Bangalore’   UPDATE Employee  SET city = 'Bangalore'  WHERE employee\_name IN (  SELECT employee\_name  FROM Works  WHERE company\_name = 'InfoSys'  );   1. Find the names, street address, and cities of residence for all employees who work for 'TechM' and earn more than $10,000.   SELECT E.employee\_name, E.street, E.city  FROM Employee AS E  JOIN Works AS W ON E.employee\_name = W.employee\_name  WHERE W.company\_name = 'TechM' AND W.salary > 10000;   1. Add Column Asset to Company table.   ALTER TABLE Company  ADD COLUMN Asset DECIMAL(15, 2); |
|  |  |
| **S6** | Consider following Relation  **Employee(emp\_id,employee\_name,street,city)**  **Works(employee\_name,company\_name,salary)**  **Company(company\_name,city)**  **Manages(employee\_name,manager\_name)**  -- Employee table  CREATE TABLE Employee (  emp\_id INT PRIMARY KEY,  employee\_name VARCHAR(50) NOT NULL,  street VARCHAR(50),  city VARCHAR(50)  );  -- Works table  CREATE TABLE Works (  employee\_name VARCHAR(50) NOT NULL,  company\_name VARCHAR(50) NOT NULL,  salary DECIMAL(10, 2) NOT NULL,  FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name),  FOREIGN KEY (company\_name) REFERENCES Company(company\_name)  );  -- Company table  CREATE TABLE Company (  company\_name VARCHAR(50) PRIMARY KEY,  city VARCHAR(50) NOT NULL  );  -- Manages table  CREATE TABLE Manages (  employee\_name VARCHAR(50) NOT NULL,  manager\_name VARCHAR(50),  FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name),  FOREIGN KEY (manager\_name) REFERENCES Employee(employee\_name)  );  Create above tables with appropriate constraints like primary key, foreign key, not null etc.   1. Change the city of employee working with InfoSys to ‘Bangalore’   UPDATE Employee  SET city = 'Bangalore'  WHERE employee\_name IN (  SELECT employee\_name  FROM Works  WHERE company\_name = 'InfoSys'  );   1. Find the names of all employees who earn more than the average salary of all employees of their company. Assume that all people work for at most one company.   SELECT employee\_name  FROM Works AS W1  WHERE salary > (  SELECT AVG(salary)  FROM Works AS W2  WHERE W1.company\_name = W2.company\_name  );   1. Find the names, street address, and cities of residence for all employees who work for 'TechM' and earn more than $10,000.   SELECT E.employee\_name, E.street, E.city  FROM Employee AS E  JOIN Works AS W ON E.employee\_name = W.employee\_name  WHERE W.company\_name = 'TechM' AND W.salary > 10000;   1. Change name of table Manages to Management.   ALTER TABLE Manages RENAME TO Management;   1. Create Simple and Unique index on employee table.   CREATE INDEX idx\_employee\_city ON Employee (city);  CREATE UNIQUE INDEX idx\_employee\_name ON Employee (employee\_name);   1. Display index Information   SHOW INDEX FROM Employee; |
|  |  |
| **P8** | Write a **Row Level Before and After Trigger** on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library\_Audit table.    CREATE TABLE Library (  book\_id INT PRIMARY KEY,  title VARCHAR(100),  author VARCHAR(100),  published\_year INT,  quantity INT  );  CREATE TABLE Library\_Audit (  audit\_id INT PRIMARY KEY AUTO\_INCREMENT,  book\_id INT,  title VARCHAR(100),  author VARCHAR(100),  published\_year INT,  quantity INT,  action VARCHAR(10), -- 'UPDATE' or 'DELETE'  action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );  CREATE OR REPLACE TRIGGER before\_update\_library  BEFORE UPDATE ON Library  FOR EACH ROW  BEGIN  -- Insert old values into Library\_Audit table before updating  INSERT INTO Library\_Audit (book\_id, title, author, published\_year, quantity, action)  VALUES (:OLD.book\_id, :OLD.title, :OLD.author, :OLD.published\_year, :OLD.quantity, 'UPDATE');  END;  /  CREATE OR REPLACE TRIGGER after\_delete\_library  AFTER DELETE ON Library  FOR EACH ROW  BEGIN  -- Insert the old values of the deleted record into Library\_Audit table  INSERT INTO Library\_Audit (book\_id, title, author, published\_year, quantity, action)  VALUES (:OLD.book\_id, :OLD.title, :OLD.author, :OLD.published\_year, :OLD.quantity, 'DELETE');  END;  / |
|  |  |
| **P9** | **Trigger**: Create a row level trigger for the CUSTOMERS table that would fire INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.  CREATE TABLE CUSTOMERS (  customer\_id INT PRIMARY KEY,  customer\_name VARCHAR(100),  salary DECIMAL(10, 2)  );  CREATE OR REPLACE TRIGGER salary\_difference\_trigger  AFTER INSERT OR UPDATE OR DELETE ON CUSTOMERS  FOR EACH ROW  DECLARE  salary\_diff DECIMAL(10, 2);  BEGIN  -- For INSERT operation, no old salary, just display message  IF INSERTING THEN  DBMS\_OUTPUT.PUT\_LINE('New customer inserted: ' || :NEW.customer\_name ||  ' with salary ' || :NEW.salary);  END IF;  -- For UPDATE operation, calculate the salary difference  IF UPDATING THEN  salary\_diff := :NEW.salary - :OLD.salary;  DBMS\_OUTPUT.PUT\_LINE('Salary updated for customer: ' || :OLD.customer\_name ||  ' from ' || :OLD.salary || ' to ' || :NEW.salary ||  '. Difference: ' || salary\_diff);  END IF;  -- For DELETE operation, show the salary before the record is deleted  IF DELETING THEN  salary\_diff := :OLD.salary; -- Display the salary before deletion  DBMS\_OUTPUT.PUT\_LINE('Customer deleted: ' || :OLD.customer\_name ||  ' with salary ' || :OLD.salary ||  '. Difference: ' || salary\_diff);  END IF;  END;  / |

|  |  |
| --- | --- |
| **S7** | Consider following Relation  **Account (Acc\_no, branch\_name,balance)**  **Branch(branch\_name,branch\_city,assets)**  **Customer(cust\_name,cust\_street,cust\_city)**  **Depositor(cust\_name,acc\_no)**  **Loan(loan\_no,branch\_name,amount)**  **Borrower(cust\_name,loan\_no)** Execute the following query:   1. Create a View1 to display List all customers in alphabetical order who have loan from Pune\_Station branch.   CREATE VIEW View1 AS  SELECT Customer.cust\_name  FROM Customer  JOIN Borrower ON Customer.cust\_name = Borrower.cust\_name  JOIN Loan ON Borrower.loan\_no = Loan.loan\_no  WHERE Loan.branch\_name = 'Pune\_Station'  ORDER BY Customer.cust\_name;   1. Create View2 on branch table by selecting any two columns and perform insert update delete operations.   CREATE VIEW View2 AS  SELECT branch\_name, branch\_city  FROM Branch;  INSERT INTO View2 (branch\_name, branch\_city) VALUES ('New\_Branch', 'New\_City');  UPDATE View2  SET branch\_city = 'Updated\_City'  WHERE branch\_name = 'New\_Branch';  DELETE FROM View2  WHERE branch\_name = 'New\_Branch';   1. Create View3 on borrower and depositor table by selecting any one column from each table perform insert update delete operations.   CREATE VIEW View3 AS  SELECT Borrower.cust\_name AS borrower\_name, Depositor.acc\_no AS depositor\_acc\_no  FROM Borrower JOIN Depositor ON Borrower.cust\_name = Depositor.cust\_name;  INSERT INTO View3 (borrower\_name, depositor\_acc\_no) VALUES ('John Doe', 12345);  UPDATE View3  SET depositor\_acc\_no = 54321  WHERE borrower\_name = 'John Doe';  DELETE FROM View3  WHERE borrower\_name = 'John Doe';   1. Create Union of left and right joint for all customers who have an account or loan or both at bank   SELECT DISTINCT cust\_name  FROM Depositor  LEFT JOIN Borrower ON Depositor.cust\_name = Borrower.cust\_name  UNION  SELECT DISTINCT cust\_name  FROM Borrower  RIGHT JOIN Depositor ON Borrower.cust\_name = Depositor.cust\_name;   1. Create Simple and Unique index.   CREATE INDEX idx\_customer\_city ON Customer (cust\_city);  CREATE UNIQUE INDEX idx\_branch\_name ON Branch (branch\_name);   1. Display index Information.   SHOW INDEX FROM Customer;  SHOW INDEX FROM Branch; |
|  |  |
| **S8** | Consider following Relation:  **Companies (comp\_id, name, cost, year) Orders (comp\_id, domain, quantity)** Execute  -- COMPANIES table  CREATE TABLE Companies (  comp\_id INT PRIMARY KEY,  name VARCHAR(50) NOT NULL,  cost DECIMAL(10, 2) NOT NULL,  year INT  );  -- ORDERS table  CREATE TABLE Orders (  comp\_id INT,  domain VARCHAR(50) NOT NULL,  quantity INT NOT NULL,  FOREIGN KEY (comp\_id) REFERENCES Companies(comp\_id)  );  -- Sample Data Insertion  INSERT INTO Companies (comp\_id, name, cost, year) VALUES  (1, 'TechCorp', 20000, 2020),  (2, 'BizWare', 15000, 2021),  (3, 'DataSolutions', 30000, 2019);  INSERT INTO Orders (comp\_id, domain, quantity) VALUES  (1, 'Software', 10),  (2, 'Hardware', 20),  (4, 'Consulting', 15);  the following query:   1. Find names, costs, domains and quantities for companies using inner join.   SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity  FROM Companies  JOIN Orders ON Companies.comp\_id = Orders.comp\_id;   1. Find names, costs, domains and quantities for companies using left outer join.   SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity  FROM Companies  LEFT JOIN Orders ON Companies.comp\_id = Orders.comp\_id;   1. Find names, costs, domains and quantities for companies using right outer join.   SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity  FROM Companies  RIGHT JOIN Orders ON Companies.comp\_id = Orders.comp\_id;   1. Find names, costs, domains and quantities for companies using Union operator.   SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity  FROM Companies  JOIN Orders ON Companies.comp\_id = Orders.comp\_id  UNION  SELECT Companies.name, Companies.cost, Orders.domain, Orders.quantity  FROM Companies  LEFT JOIN Orders ON Companies.comp\_id = Orders.comp\_id  WHERE Orders.comp\_id IS NULL;   1. Create View View1 by selecting both tables to show company name and quantities.   CREATE VIEW View1 AS  SELECT Companies.name, Orders.quantity  FROM Companies  JOIN Orders ON Companies.comp\_id = Orders.comp\_id;   1. Create View View2 by selecting any two columns and perform insert update delete operations.   -- Creating View2  CREATE VIEW View2 AS  SELECT name, cost  FROM Companies;  INSERT INTO View2 (name, cost) VALUES ('NewCo', 18000);  UPDATE View2  SET cost = 22000  WHERE name = 'NewCo';  DELETE FROM View2  WHERE name = 'NewCo';   1. Display content of View1, View2.   SELECT \* FROM View1;  SELECT \* FROM View2; |
|  |  |
| **P5** | Write a **PL/SQL Block** to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, whenever such salary updates take place, a record for same is maintained in the increment\_salary table.  emp(emp\_no, salary) increment\_salary(emp\_no, salary)  -- Create the emp table  CREATE TABLE emp (  emp\_no INT PRIMARY KEY,  salary DECIMAL(10, 2)  );  -- Create the increment\_salary table to store salary increments  CREATE TABLE increment\_salary (  emp\_no INT,  salary DECIMAL(10, 2),  increment\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );    DECLARE  -- Declare a variable to store the average salary of all employees  avg\_salary DECIMAL(10, 2);    -- Cursor to fetch employees with salary less than average salary  CURSOR emp\_cursor IS  SELECT emp\_no, salary  FROM emp  WHERE salary < avg\_salary;    -- Declare a variable to store the new salary after increment  new\_salary DECIMAL(10, 2);    BEGIN  -- Calculate the average salary in the organization  SELECT AVG(salary) INTO avg\_salary FROM emp;    -- Loop through each employee who earns less than the average salary  FOR emp\_rec IN emp\_cursor LOOP  -- Calculate the new salary (10% increase)  new\_salary := emp\_rec.salary \* 1.10;    -- Update the employee's salary in the emp table  UPDATE emp  SET salary = new\_salary  WHERE emp\_no = emp\_rec.emp\_no;    -- Insert the old salary and new salary into increment\_salary table  INSERT INTO increment\_salary (emp\_no, salary)  VALUES (emp\_rec.emp\_no, new\_salary);    -- Optionally, print a message for each update  DBMS\_OUTPUT.PUT\_LINE('Employee ' || emp\_rec.emp\_no || ' salary updated to ' || new\_salary);  END LOOP;  -- Commit the changes to the database  COMMIT;    EXCEPTION  WHEN OTHERS THEN  -- If an error occurs, rollback the transaction and print error  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END;  / |

|  |  |
| --- | --- |
| **S9** | **SQL Queries**  Create following tables with suitable constraints. Insert data and solve the following queries:  **CUSTOMERS(CNo, Cname, Ccity, CMobile)**  **ITEMS(INo, Iname, Itype, Iprice, Icount)**  **PURCHASE(PNo, Pdate, Pquantity, Cno, INo)**  -- CUSTOMERS table  CREATE TABLE CUSTOMERS (  CNo INT PRIMARY KEY,  Cname VARCHAR(50) NOT NULL,  Ccity VARCHAR(50),  CMobile VARCHAR(15) UNIQUE  );  -- ITEMS table  CREATE TABLE ITEMS (  INo INT PRIMARY KEY,  Iname VARCHAR(50) NOT NULL,  Itype VARCHAR(50) NOT NULL,  Iprice DECIMAL(10, 2) NOT NULL,  Icount INT NOT NULL  );  -- PURCHASE table  CREATE TABLE PURCHASE (  PNo INT PRIMARY KEY,  Pdate DATE NOT NULL,  Pquantity INT NOT NULL,  CNo INT,  INo INT,  FOREIGN KEY (CNo) REFERENCES CUSTOMERS(CNo),  FOREIGN KEY (INo) REFERENCES ITEMS(INo)  );  -- Sample Data Insertion  INSERT INTO CUSTOMERS (CNo, Cname, Ccity, CMobile) VALUES  (1, 'Gopal', 'Mumbai', '1234567890'),  (2, 'Maya', 'Pune', '0987654321'),  (3, 'Amit', 'Delhi', '1122334455');  INSERT INTO ITEMS (INo, Iname, Itype, Iprice, Icount) VALUES  (1, 'Notebook', 'Stationary', 500, 100),  (2, 'Pen', 'Stationary', 50, 500),  (3, 'Calculator', 'Electronics', 1500, 50),  (4, 'Marker', 'Stationary', 300, 200);  INSERT INTO PURCHASE (PNo, Pdate, Pquantity, CNo, INo) VALUES  (1, '2023-10-01', 2, 1, 1),  (2, '2023-11-05', 1, 2, 3),  (3, '2023-09-15', 5, 2, 2);   1. List all stationary items with price between 400/- to 1000/-   SELECT \*  FROM ITEMS  WHERE Itype = 'Stationary' AND Iprice BETWEEN 400 AND 1000;   1. Change the mobile number of customer “Gopal”   UPDATE CUSTOMERS  SET CMobile = '9876543210'  WHERE Cname = 'Gopal';   1. Display the item with maximum price   SELECT \*  FROM ITEMS  WHERE Iprice = (SELECT MAX(Iprice) FROM ITEMS);   1. Display all purchases sorted from the most recent to the oldest   SELECT \*  FROM PURCHASE  ORDER BY Pdate DESC;   1. Count the number of customers in every city   SELECT Ccity, COUNT(\*) AS num\_of\_customers  FROM CUSTOMERS  GROUP BY Ccity;   1. Display all purchased quantity of Customer Maya   SELECT Pquantity  FROM PURCHASE  JOIN CUSTOMERS ON PURCHASE.CNo = CUSTOMERS.CNo  WHERE CUSTOMERS.Cname = 'Maya';   1. Create view which shows Iname, Price and Count of all stationary items in descending order of price.   CREATE VIEW StationaryItemsView AS  SELECT Iname, Iprice AS Price, Icount AS Count  FROM ITEMS  WHERE Itype = 'Stationary'  ORDER BY Iprice DESC; |
|  |  |
| **M1** | **Design and Develop MongoDB Queries using CRUD operations:**  Create Employee collection by considering following Fields:   1. Name: Embedded Doc (FName, LName) 2. Company Name: String 3. Salary: Number 4. Designation: String 5. Age: Number 6. Expertise: Array 7. DOB: String or Date 8. Email id: String ix. Contact: String   x. Address: Array of Embedded Doc (PAddr, LAddr)  Insert at least 5 documents in collection by considering above attribute and execute following queries:  db.Employee.insertMany([  {  "Name": { "FName": "John", "LName": "Doe" },  "Company Name": "TCS",  "Salary": 50000,  "Designation": "Programmer",  "Age": 28,  "Expertise": ["Java", "Spring", "MongoDB"],  "DOB": "1995-06-15",  "Email id": "john.doe@tcs.com",  "Contact": "9876543210",  "Address": [  { "PAddr": "123 Main St", "LAddr": "Mumbai" }  ]  },  {  "Name": { "FName": "Jane", "LName": "Smith" },  "Company Name": "Infosys",  "Salary": 45000,  "Designation": "Tester",  "Age": 24,  "Expertise": ["Testing", "Automation", "Selenium"],  "DOB": "1999-04-12",  "Email id": "jane.smith@infosys.com",  "Contact": "9871234567",  "Address": [  { "PAddr": "456 Elm St", "LAddr": "Pune" }  ]  },  {  "Name": { "FName": "Alice", "LName": "Johnson" },  "Company Name": "Infosys",  "Salary": 55000,  "Designation": "Programmer",  "Age": 30,  "Expertise": ["Java", "Python", "C++"],  "DOB": "1993-05-20",  "Email id": "alice.johnson@infosys.com",  "Contact": "5559876543",  "Address": [  { "PAddr": "789 Oak St", "LAddr": "Chennai" }  ]  },  {  "Name": { "FName": "Bob", "LName": "Williams" },  "Company Name": "TCS",  "Salary": 60000,  "Designation": "Manager",  "Age": 35,  "Expertise": ["Project Management", "Agile", "Scrum"],  "DOB": "1988-11-30",  "Email id": "bob.williams@tcs.com",  "Contact": "7776543210",  "Address": [  { "PAddr": "123 Pine St", "LAddr": "Bangalore" }  ]  },  {  "Name": { "FName": "Eve", "LName": "Davis" },  "Company Name": "TCS",  "Salary": 70000,  "Designation": "Programmer",  "Age": 27,  "Expertise": ["Java", "Spring Boot", "Microservices"],  "DOB": "1996-03-15",  "Email id": "eve.davis@tcs.com",  "Contact": "6669876543",  "Address": [  { "PAddr": "234 Maple St", "LAddr": "Chennai" }  ]  }  ]);   1. Select all documents where the Designation field has the value "Programmer" and the value of the salary field is greater than 30000.   db.Employee.find(  {  "Designation": "Programmer",  "Salary": { $gt: 30000 }  }  );   1. Creates a new document if no document in the employee collection contains   {Designation: "Tester", Company\_name: "TCS", Age: 25}  var existingDoc = db.Employee.findOne({ "Designation": "Tester", "Company Name": "TCS", "Age": 25 });  if (!existingDoc) {  db.Employee.insertOne({  "Name": { "FName": "Sam", "LName": "Taylor" },  "Company Name": "TCS",  "Salary": 40000,  "Designation": "Tester",  "Age": 25,  "Expertise": ["Automation", "Selenium", "JMeter"],  "DOB": "1998-01-10",  "Email id": "sam.taylor@tcs.com",  "Contact": "9876543212",  "Address": [  { "PAddr": "123 Park Ave", "LAddr": "Pune" }  ]  });  }   1. Increase salary of each Employee working with “Infosys" 10000.   db.Employee.updateMany(  { "Company Name": "Infosys" },  { $inc: { "Salary": 10000 } }  );   1. Finds all employees working with "TCS" and reduce their salary by 5000.   db.Employee.updateMany(  { "Company Name": "TCS" },  { $inc: { "Salary": -5000 } }  );   1. Return documents where Designation is not equal to "Tester".   db.Employee.find(  { "Designation": { $ne: "Tester" } }  );   1. Find all employee with Exact Match on an Array having Expertise:   ['Mongodb','Mysql','Cassandra']  db.Employee.find(  { "Expertise": ["Mongodb", "Mysql", "Cassandra"] }  ); |
|  |  |

|  |  |
| --- | --- |
| **M2** | **Design and Develop MongoDB Queries using CRUD operations:**  Create Employee collection by considering following Fields:   1. Name: Embedded Doc (FName, LName) 2. Company Name: String 3. Salary: Number 4. Designation: String 5. Age: Number 6. Expertise: Array 7. DOB: String or Date 8. Email id: String ix. Contact: String   x. Address: Array of Embedded Doc (PAddr, LAddr)  Insert at least 5 documents in collection by considering above attribute and execute following queries:  db.Employee.insertMany([  {  "Name": { "FName": "John", "LName": "Doe" },  "Company Name": "TCS",  "Salary": 120000,  "Designation": "Developer",  "Age": 28,  "Expertise": ["Java", "Spring", "Hibernate"],  "DOB": "1995-08-10",  "Email id": "john.doe@tcs.com",  "Contact": "9876543210",  "Address": [  { "PAddr": "123 Main St", "LAddr": "Mumbai" }  ]  },  {  "Name": { "FName": "Jane", "LName": "Smith" },  "Company Name": "Infosys",  "Salary": 95000,  "Designation": "Tester",  "Age": 24,  "Expertise": ["Testing", "Selenium", "Automation"],  "DOB": "1999-06-15",  "Email id": "jane.smith@infosys.com",  "Contact": "9871234567",  "Address": [  { "PAddr": "456 Elm St", "LAddr": "Pune" }  ]  },  {  "Name": { "FName": "Alice", "LName": "Johnson" },  "Company Name": "TCS",  "Salary": 80000,  "Designation": "Developer",  "Age": 32,  "Expertise": ["Python", "Machine Learning"],  "DOB": "1991-05-25",  "Email id": "alice.johnson@tcs.com",  "Contact": "5559876543",  "Address": [  { "PAddr": "789 Oak St", "LAddr": "Delhi" }  ]  },  {  "Name": { "FName": "Bob", "LName": "Williams" },  "Company Name": "Wipro",  "Salary": 105000,  "Designation": "Manager",  "Age": 35,  "Expertise": ["Project Management", "Agile"],  "DOB": "1988-11-30",  "Email id": "bob.williams@wipro.com",  "Contact": "7776543210",  "Address": [  { "PAddr": "123 Pine St", "LAddr": "Bangalore" }  ]  },  {  "Name": { "FName": "Eve", "LName": "Davis" },  "Company Name": "Infosys",  "Salary": 115000,  "Designation": "Developer",  "Age": 26,  "Expertise": ["Java", "Spring Boot", "Microservices"],  "DOB": "1997-03-21",  "Email id": "eve.davis@infosys.com",  "Contact": "6669876543",  "Address": [  { "PAddr": "234 Maple St", "LAddr": "Chennai" }  ]  }  ]);   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": 1, "\_id": 0 }  );   1. Creates a new document if no document in the employee collection contains   {Designation: "Tester", Company\_name: "TCS", Age: 25}  db.Employee.find(  { "Age": { $lt: 30 }, "Salary": { $gt: 50000 } },  { "Name": 1, "\_id": 0 }  );  var existingDoc = db.Employee.findOne({ "Designation": "Tester", "Company Name": "TCS", "Age": 25 });  if (!existingDoc) {  db.Employee.insertOne({  "Name": { "FName": "Sam", "LName": "Taylor" },  "Company Name": "TCS",  "Salary": 90000,  "Designation": "Tester",  "Age": 25,  "Expertise": ["Automation", "Selenium", "TestNG"],  "DOB": "1998-02-15",  "Email id": "sam.taylor@tcs.com",  "Contact": "9876543211",  "Address": [  { "PAddr": "123 Park Ave", "LAddr": "Pune" }  ]  });  }   1. Selects all documents in the collection where the field age has a value less than 30 or the value of the salary field is greater than 40000.   db.Employee.find({  $or: [  { "Age": { $lt: 30 } },  { "Salary": { $gt: 40000 } }  ]  });   1. Find documents where Designation is not equal to "Developer".   db.Employee.find({ "Designation": { $ne: "Developer" } });   1. Find \_id, Designation, Address and Name from all documents where Company\_name is "Infosys".   db.Employee.find(  { "Company Name": "Infosys" },  { "\_id": 1, "Designation": 1, "Address": 1, "Name": 1 }  );   1. Display only FName and LName of all Employees   db.Employee.find(  {},  { "Name.FName": 1, "Name.LName": 1, "\_id": 0 }  ); |
|  |  |
| **P4** | Write a **PL/SQL block** for following requirements and handle the exceptions. Roll no. of students will be entered by the user. Attendance of roll no. entered by user will be checked in the Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “Detained”. Otherwise display message “Term granted” and set the status in stud table as “Not Detained”. **Student (Roll, Name, Attendance, Status)**    CREATE TABLE stud (  roll INT PRIMARY KEY,  name VARCHAR(100),  attendance DECIMAL(5, 2), -- Attendance percentage  status VARCHAR(20) -- Status: Detained / Not Detained  );  DECLARE  -- Variable to store the roll number entered by the user  v\_roll INT;    -- Variable to store the attendance of the student  v\_attendance DECIMAL(5, 2);    -- Variable to store the current student's status  v\_status VARCHAR(20);    BEGIN  -- Prompt the user to input the roll number  v\_roll := &roll\_no; -- This will prompt the user to enter the roll number    -- Fetch the student's attendance from the stud table  BEGIN  SELECT attendance, status  INTO v\_attendance, v\_status  FROM stud  WHERE roll = v\_roll;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  -- Handle the case when no student with the given roll number is found  DBMS\_OUTPUT.PUT\_LINE('No student found with roll number ' || v\_roll);  RAISE; -- Raise exception to exit the block  WHEN OTHERS THEN  -- Handle any other exceptions  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  RAISE; -- Raise exception to exit the block  END;    -- Check the attendance and update the status accordingly  IF v\_attendance < 75 THEN  -- If attendance is less than 75%, set status to "Detained"  DBMS\_OUTPUT.PUT\_LINE('Term not granted');  UPDATE stud  SET status = 'Detained'  WHERE roll = v\_roll;  ELSE  -- If attendance is 75% or more, set status to "Not Detained"  DBMS\_OUTPUT.PUT\_LINE('Term granted');  UPDATE stud  SET status = 'Not Detained'  WHERE roll = v\_roll;  END IF;    -- Commit the changes to the database  COMMIT;    EXCEPTION  WHEN OTHERS THEN  -- In case of any error, rollback the transaction  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END;  / |
|  |  |
| **P7** | Create a **stored function** titled **'Age\_calc'**.  Accept the date of birth of a person as a parameter.  Calculate the age of the person in years, months and days e.g. 3 years, 2months, 10 days.  Return the age in years directly (with the help of Return statement). The months and days are to be returned indirectly in the form of OUT parameters.    CREATE OR REPLACE FUNCTION Age\_calc (  dob IN DATE, -- Date of birth of the person  months OUT NUMBER, -- The calculated months, returned as OUT parameter  days OUT NUMBER -- The calculated days, returned as OUT parameter  )  RETURN NUMBER -- The age in years  IS  v\_age\_in\_years NUMBER; -- Variable to store the age in years  v\_age\_in\_months NUMBER; -- Variable to store the age in months  v\_age\_in\_days NUMBER; -- Variable to store the age in days  BEGIN  -- Calculate the age in months from the date of birth till the current date  v\_age\_in\_months := MONTHS\_BETWEEN(SYSDATE, dob);    -- Calculate the years part (truncate months to years)  v\_age\_in\_years := TRUNC(v\_age\_in\_months / 12);    -- Calculate remaining months after extracting the years  months := TRUNC(v\_age\_in\_months) - (v\_age\_in\_years \* 12);    -- Calculate remaining days by subtracting the full years and months from the current date  v\_age\_in\_days := SYSDATE - (ADD\_MONTHS(dob, v\_age\_in\_years \* 12 + months));  days := TRUNC(v\_age\_in\_days);    -- Return the age in years directly  RETURN v\_age\_in\_years;    EXCEPTION  WHEN OTHERS THEN  -- Handle exceptions and return 0 for error case  RETURN 0;  END Age\_calc;  / |

|  |  |
| --- | --- |
| **M3** | **Design and Develop MongoDB Queries using CRUD operations:**  Create Employee collection by considering following Fields: i. Emp\_id : Number   1. Name: Embedded Doc (FName, LName) 2. Company Name: String 3. Salary: Number 4. Designation: String 5. Age: Number 6. Expertise: Array 7. DOB: String or Date 8. Email id: String x. Contact: String   xi. Address: Array of Embedded Doc (PAddr, LAddr)  Insert at least 5 documents in collection by considering above attribute and execute following queries:  db.Employee.insertMany([  {  "Emp\_id": 1001,  "Name": { "FName": "John", "LName": "Doe" },  "Company Name": "TCS",  "Salary": 120000,  "Designation": "Tester",  "Age": 25,  "Expertise": ["Automation", "Testing", "Java"],  "DOB": "1998-01-15",  "Email id": "john.doe@tcs.com",  "Contact": "987-654-3210",  "Address": [  { "PAddr": "123 Main St", "LAddr": "Pune", "Pin\_code": "411001" }  ]  },  {  "Emp\_id": 1002,  "Name": { "FName": "Jane", "LName": "Smith" },  "Company Name": "TCS",  "Salary": 100000,  "Designation": "Developer",  "Age": 30,  "Expertise": ["NodeJS", "MongoDB", "JavaScript"],  "DOB": "1993-07-20",  "Email id": "jane.smith@tcs.com",  "Contact": "987-123-4567",  "Address": [  { "PAddr": "456 Elm St", "LAddr": "Mumbai", "Pin\_code": "400001" }  ]  },  {  "Emp\_id": 1003,  "Name": { "FName": "Bob", "LName": "Williams" },  "Company Name": "Wipro",  "Salary": 140000,  "Designation": "Tester",  "Age": 28,  "Expertise": ["Manual Testing", "SQL", "Agile"],  "DOB": "1995-05-30",  "Email id": "bob.williams@wipro.com",  "Contact": "666-432-9876",  "Address": [  { "PAddr": "789 Oak St", "LAddr": "Bangalore", "Pin\_code": "560001" }  ]  },  {  "Emp\_id": 1004,  "Name": { "FName": "Alice", "LName": "Johnson" },  "Company Name": "TCS",  "Salary": 110000,  "Designation": "Developer",  "Age": 26,  "Expertise": ["Java", "Spring", "Hibernate"],  "DOB": "1997-08-10",  "Email id": "alice.johnson@tcs.com",  "Contact": "555-555-5555",  "Address": [  { "PAddr": "321 Pine St", "LAddr": "Pune", "Pin\_code": "411001" }  ]  },  {  "Emp\_id": 1005,  "Name": { "FName": "Eve", "LName": "Davis" },  "Company Name": "Accenture",  "Salary": 130000,  "Designation": "Manager",  "Age": 35,  "Expertise": ["Project Management", "Leadership", "Agile"],  "DOB": "1988-03-20",  "Email id": "eve.davis@accenture.com",  "Contact": "111-222-3333",  "Address": [  { "PAddr": "500 Maple St", "LAddr": "Chennai", "Pin\_code": "600001" }  ]  }  ]);   1. Creates a new document if no document in the employee collection contains   {Designation: "Tester", Company\_name: "TCS", Age: 25}  var existingDoc = db.Employee.findOne({ "Designation": "Tester", "Company Name": "TCS", "Age": 25 });  if (!existingDoc) {  db.Employee.insertOne({  "Emp\_id": 1006,  "Name": { "FName": "Sam", "LName": "Taylor" },  "Company Name": "TCS",  "Salary": 90000,  "Designation": "Tester",  "Age": 25,  "Expertise": ["Automation", "Selenium", "Testing"],  "DOB": "1998-03-25",  "Email id": "sam.taylor@tcs.com",  "Contact": "123-321-4321",  "Address": [  { "PAddr": "777 Park Ave", "LAddr": "Pune", "Pin\_code": "411001" }  ]  });  }   1. Finds all employees working with Company\_name: "TCS" and increase their salary by 2000.   db.Employee.updateMany(  { "Company Name": "TCS" },  { $inc: { "Salary": 2000 } }  );   1. Matches all documents where the value of the field Address is an embedded document that contains only the field city with the value "Pune" and the field Pin\_code with the value "411001".   db.Employee.find({  "Address": {  $elemMatch: {  "LAddr": "Pune",  "Pin\_code": "411001"  }  }  });   1. Find employee details who are working as "Developer" or "Tester".   db.Employee.find({  "Designation": { $in: ["Developer", "Tester"] }  });   1. Drop Single documents where designation="Developer".   db.Employee.deleteOne({ "Designation": "Developer" });   1. Count number of documents in employee collection.   db.Employee.countDocuments(); |
|  |  |
| **M6** | **Design MongoDB database and perform following Map reduce operation:**  Create Employee collection by considering following Fields:   1. Name: Embedded Doc (FName, LName) 2. Company Name: String 3. Salary: Number 4. Designation: String 5. Age: Number 6. Expertise: Array 7. DOB: String or Date 8. Email id: String ix. Contact: String   x. Address: Array of Embedded Doc (PAddr, LAddr) Execute the following query:  {  "Name": {  "FName": "John",  "LName": "Doe"  },  "Company Name": "TCS",  "Salary": 120000,  "Designation": "DBA",  "Age": 32,  "Expertise": ["SQL", "MongoDB", "Database Management"],  "DOB": "1991-03-15",  "Email id": "john.doe@example.com",  "Contact": "123-456-7890",  "Address": [  { "PAddr": "123 Main St", "LAddr": "Pune" }  ]  }   1. Display the total salary of per company   var map = function() {  emit(this['Company Name'], this['Salary']);  };  var reduce = function(key, values) {  return Array.sum(values);  };  db.Employee.mapReduce(map, reduce, { out: "total\_salary\_per\_company" });   1. Display the total salary of company Name:"TCS"   var map = function() {  if (this['Company Name'] == 'TCS') {  emit(this['Company Name'], this['Salary']);  }  };  var reduce = function(key, values) {  return Array.sum(values);  };  db.Employee.mapReduce(map, reduce, { out: "total\_salary\_tcs" });   1. Return the average salary of company whose address is “Pune".   var map = function() {  for (var i = 0; i < this.Address.length; i++) {  if (this.Address[i].LAddr == 'Pune') {  emit(this['Company Name'], { salary: this['Salary'], count: 1 });  }  }  };  var reduce = function(key, values) {  var totalSalary = 0;  var totalCount = 0;  values.forEach(function(value) {  totalSalary += value.salary;  totalCount += value.count;  });  return { salary: totalSalary, count: totalCount };  };  var finalize = function(key, reducedValue) {  reducedValue.averageSalary = reducedValue.salary / reducedValue.count;  return reducedValue;  };  db.Employee.mapReduce(map, reduce, { out: "average\_salary\_pune", finalize: finalize });   1. Display total count for “City=Pune”   var map = function() {  for (var i = 0; i < this.Address.length; i++) {  if (this.Address[i].LAddr == 'Pune') {  emit('Pune', 1);  }  }  };  var reduce = function(key, values) {  return Array.sum(values);  };  db.Employee.mapReduce(map, reduce, { out: "count\_pune" });   1. Return count for city pune and age greater than 40.   var map = function() {  for (var i = 0; i < this.Address.length; i++) {  if (this.Address[i].LAddr == 'Pune' && this['Age'] > 40) {  emit('Pune\_Above\_40', 1);  }  }  };  var reduce = function(key, values) {  return Array.sum(values);  };  db.Employee.mapReduce(map, reduce, { out: "count\_pune\_above\_40" }); |

|  |  |
| --- | --- |
| **M4** | **Design and Develop MongoDB Queries using Aggregation operations:**  Create Employee collection by considering following Fields: i. Emp\_id : Number   1. Name: Embedded Doc (FName, LName) 2. Company Name: String 3. Salary: Number 4. Designation: String 5. Age: Number 6. Expertise: Array 7. DOB: String or Date 8. Email id: String x. Contact: String   xi. Address: Array of Embedded Doc (PAddr, LAddr)  db.Employee.insertMany([  {  "Emp\_id": 1001,  "Name": { "FName": "John", "LName": "Doe" },  "Company Name": "TechCorp",  "Salary": 120000,  "Designation": "DBA",  "Age": 32,  "Expertise": ["SQL", "MongoDB", "Database Management"],  "DOB": ISODate("1991-03-15"),  "Email id": "john.doe@example.com",  "Contact": "123-456-7890",  "Address": [  { "PAddr": "123 Main St", "LAddr": "Cityville" },  { "PAddr": "456 Elm St", "LAddr": "Townsville" }  ]  },  {  "Emp\_id": 1002,  "Name": { "FName": "Jane", "LName": "Smith" },  "Company Name": "WebSoft",  "Salary": 200000,  "Designation": "DBA",  "Age": 30,  "Expertise": ["NoSQL", "MongoDB", "DBA"],  "DOB": ISODate("1993-07-20"),  "Email id": "jane.smith@example.com",  "Contact": "987-654-3210",  "Address": [  { "PAddr": "789 Oak St", "LAddr": "Lakeside" },  { "PAddr": "321 Pine St", "LAddr": "Riverside" }  ]  },  {  "Emp\_id": 1003,  "Name": { "FName": "Swapnil", "LName": "Jadhav" },  "Company Name": "TechCorp",  "Salary": 180000,  "Designation": "Engineer",  "Age": 28,  "Expertise": ["Python", "MongoDB"],  "DOB": ISODate("1995-01-10"),  "Email id": "swapnil.jadhav@example.com",  "Contact": "555-789-1234",  "Address": [  { "PAddr": "500 Maple St", "LAddr": "Cityville" },  { "PAddr": "700 Cedar St", "LAddr": "Uptown" }  ]  },  {  "Emp\_id": 1004,  "Name": { "FName": "Amit", "LName": "Patel" },  "Company Name": "WebSoft",  "Salary": 250000,  "Designation": "DBA",  "Age": 34,  "Expertise": ["MySQL", "MongoDB", "DBA"],  "DOB": ISODate("1989-05-05"),  "Email id": "amit.patel@example.com",  "Contact": "666-432-9876",  "Address": [  { "PAddr": "123 Elm St", "LAddr": "Woodland" },  { "PAddr": "900 Birch St", "LAddr": "Hillside" }  ]  },  {  "Emp\_id": 1005,  "Name": { "FName": "Emily", "LName": "White" },  "Company Name": "TechCorp",  "Salary": 220000,  "Designation": "Manager",  "Age": 40,  "Expertise": ["Leadership", "Project Management"],  "DOB": ISODate("1983-11-15"),  "Email id": "emily.white@example.com",  "Contact": "888-555-1234",  "Address": [  { "PAddr": "100 Pine St", "LAddr": "Mountainview" },  { "PAddr": "400 Oak St", "LAddr": "Seaside" }  ]  }  ]);Insert at least 5 documents in collection by considering above attribute and execute following:   1. Using aggregation Return Designation with Total Salary is Above 200000.   db.Employee.aggregate([  { $group: { \_id: "$Designation", totalSalary: { $sum: "$Salary" } } },  { $match: { totalSalary: { $gt: 200000 } } }  ]);   1. Using Aggregate method returns names and \_id in upper case and in alphabetical order.   db.Employee.aggregate([  { $project: {  Name: { $concat: [{ $toUpper: "$Name.FName" }, " ", { $toUpper: "$Name.LName" }] },  \_id: 1  }  },  { $sort: { Name: 1 } }  ]);   1. Using aggregation method find Employee with Total Salary for Each City with Designation="DBA".   db.Employee.aggregate([  { $match: { Designation: "DBA" } },  { $unwind: "$Address" },  { $group: { \_id: "$Address.LAddr", totalSalary: { $sum: "$Salary" } } }  ]);   1. Create Single Field Indexes on Designation field of employee collection   db.Employee.createIndex({ "Designation": 1 });   1. To Create Multikey Indexes on Expertise field of employee collection.   db.Employee.createIndex({ "Expertise": 1 });   1. Create an Index on Emp\_id field, compare the time require to search Emp\_id before and after creating an index. (Hint Add at least 10000 Documents)   let start = new Date();  db.Employee.find({ "Emp\_id": 5000 });  let end = new Date();  print("Time before index: " + (end - start) + " ms");  db.Employee.createIndex({ "Emp\_id": 1 });  start = new Date();  db.Employee.find({ "Emp\_id": 5000 });  end = new Date();  print("Time after index: " + (end - start) + " ms");   1. Return a List of Indexes on created on employee Collection.   db.Employee.getIndexes(); |
|  |  |
| **P6** | Write a **Stored Procedure** namely **proc\_Grade** for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and 900 categories is first class, if marks 899 and 825 category is Higher Second Class.  Write a PL/SQL block for using procedure created with above requirement.  Stud\_Marks(name, total\_marks),  Result (Roll,Name, Class)  CREATE TABLE Stud\_Marks (  name VARCHAR2(50) NOT NULL, -- Student name  total\_marks NUMBER NOT NULL, -- Total marks scored by the student  CONSTRAINT pk\_stud\_marks PRIMARY KEY (name) -- Primary key constraint  );  CREATE TABLE Result (  Roll NUMBER PRIMARY KEY, -- Unique roll number  Name VARCHAR2(50) NOT NULL, -- Student name  Class VARCHAR2(50) NOT NULL -- Categorized class (Distinction, First Class, etc.)  );    CREATE OR REPLACE PROCEDURE proc\_Grade (  p\_name IN VARCHAR2, -- Name of the student  p\_marks IN NUMBER -- Total marks scored by the student  )  IS  v\_class VARCHAR2(20); -- Variable to hold the category/class of the student  BEGIN  -- Categorization based on marks  IF p\_marks >= 990 AND p\_marks <= 1500 THEN  v\_class := 'Distinction';  ELSIF p\_marks >= 900 AND p\_marks <= 989 THEN  v\_class := 'First Class';  ELSIF p\_marks >= 825 AND p\_marks <= 899 THEN  v\_class := 'Higher Second Class';  ELSE  v\_class := 'Not Categorized'; -- In case marks don't fit into any category  END IF;  -- Insert the result into Result table  INSERT INTO Result (Name, Class)  VALUES (p\_name, v\_class);  COMMIT; -- Commit the transaction  EXCEPTION  WHEN OTHERS THEN  -- Handle any exceptions (e.g., insertion failure)  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error occurred while categorizing the student.');  END proc\_Grade;  / |
|  |  |
| **C1** | Write a program to implement **MongoDB database connectivity** with PHP /python /Java Implement Database navigation CRUD operations (add, delete, edit etc.)    composer require mongodb/mongodb  <?php  require 'vendor/autoload.php'; // Autoload the MongoDB library  // Create MongoDB Client  $client = new MongoDB\Client("mongodb://localhost:27017");  // Select the database and collection  $collection = $client->mydatabase->employees;  // CREATE - Insert a new employee  $insertResult = $collection->insertOne([  'name' => ['first' => 'John', 'last' => 'Doe'],  'age' => 28,  'company' => 'TCS',  'salary' => 50000,  'designation' => 'Programmer',  ]);  echo "Inserted employee with ID: " . $insertResult->getInsertedId() . "\n";  // READ - Find all employees  $employees = $collection->find();  foreach ($employees as $employee) {  echo "Name: " . $employee['name']['first'] . " " . $employee['name']['last'] . "\n";  }  // UPDATE - Update salary of employee where name is 'John'  $updateResult = $collection->updateOne(  ['name.first' => 'John'], // Filter  ['$set' => ['salary' => 55000]] // Update  );  echo "Matched " . $updateResult->getMatchedCount() . " document(s)\n";  // DELETE - Delete employee with name 'John'  $deleteResult = $collection->deleteOne(['name.first' => 'John']);  echo "Deleted " . $deleteResult->getDeletedCount() . " document(s)\n";  ?> |
|  |  |

|  |  |
| --- | --- |
| **C2** | Implement **MYSQL/Oracle database connectivity** with PHP /python /Java  Implement Database navigation operations (add, delete, edit,).  **PHP (MySQL)**  **Step 1: MySQL Database Connection in PHP**  php  Copy code  <?php  $servername = "localhost";  $username = "root";  $password = "";  $dbname = "your\_database\_name";  // Create connection  $conn = new mysqli($servername, $username, $password, $dbname);  // Check connection  if ($conn->connect\_error) {  die("Connection failed: " . $conn->connect\_error);  }  echo "Connected successfully";  ?>  **Step 2: Database Navigation (Add, Delete, Edit)**   * **Add (Insert)**:   php  Copy code  <?php  $sql = "INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com')";  if ($conn->query($sql) === TRUE) {  echo "New record created successfully";  } else {  echo "Error: " . $sql . "<br>" . $conn->error;  }  ?>   * **Delete**:   php  Copy code  <?php  $sql = "DELETE FROM users WHERE id = 1";  if ($conn->query($sql) === TRUE) {  echo "Record deleted successfully";  } else {  echo "Error: " . $sql . "<br>" . $conn->error;  }  ?>   * **Edit (Update)**:   php  Copy code  <?php  $sql = "UPDATE users SET email='newemail@example.com' WHERE id=1";  if ($conn->query($sql) === TRUE) {  echo "Record updated successfully";  } else {  echo "Error: " . $sql . "<br>" . $conn->error;  }  ?>  **Step 3: Closing the Connection**  php  Copy code  $conn->close();  ?> |
|  |  |
| **P2** | Write an **Unnamed PL/SQL** of code for the following requirements: -  Schema:  Borrower (Rollin, Name, DateofIssue, NameofBook, Status)  Fine (Roll\_no,Date,Amt)  Accept roll\_no & name of book from user.  Check the number of days (from date of issue).   1. If days are between 15 to 30 then fine amounts will be Rs 5 per day. 2. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day. 3. After submitting the book, status will change from I to R. 4. If condition of fine is true, then details will be stored into fine table.   -- Create the Borrower table to store information about borrowed books  CREATE TABLE Borrower (  Rollin NUMBER PRIMARY KEY, -- Roll number of the student (Primary Key)  Name VARCHAR2(100) NOT NULL, -- Name of the student  DateofIssue DATE NOT NULL, -- Date when the book was issued  NameofBook VARCHAR2(255) NOT NULL, -- Name of the borrowed book  Status CHAR(1) CHECK (Status IN ('I', 'R')) -- Status of the book (I = Issued, R = Returned)  );  -- Create the Fine table to store the fine details  CREATE TABLE Fine (  Roll\_no NUMBER, -- Roll number of the student  Date DATE DEFAULT SYSDATE, -- Date when the fine is registered  Amt NUMBER(10, 2), -- Amount of the fine  CONSTRAINT fk\_rollno FOREIGN KEY (Roll\_no) REFERENCES Borrower(Rollin) -- Foreign Key referencing Borrower table  );  DECLARE  v\_roll\_no NUMBER; -- Variable to hold roll number  v\_name VARCHAR2(100); -- Variable to hold name of the book  v\_dateofissue DATE; -- Variable to hold date of issue of the book  v\_days NUMBER; -- Variable to calculate number of days  v\_fine\_amt NUMBER := 0; -- Variable to hold the fine amount  v\_status CHAR(1); -- Variable to hold the current status of the book  BEGIN  -- Accept roll\_no and name of the book as inputs from the user  v\_roll\_no := &roll\_no;  v\_name := '&book\_name';    -- Retrieve the DateofIssue and Status for the provided roll\_no and NameofBook  SELECT DateofIssue, Status INTO v\_dateofissue, v\_status  FROM Borrower  WHERE Rollin = v\_roll\_no AND NameofBook = v\_name;    -- Calculate the number of days from DateofIssue to today's date  v\_days := TRUNC(SYSDATE) - v\_dateofissue;    -- If days between 15 and 30, fine will be Rs. 5 per day  IF v\_days BETWEEN 15 AND 30 THEN  v\_fine\_amt := v\_days \* 5;    -- If days are greater than 30, fine is Rs. 50 per day for first 30 days and Rs. 5 per day after  ELSIF v\_days > 30 THEN  v\_fine\_amt := (30 \* 50) + ((v\_days - 30) \* 5);  END IF;  -- If there is a fine, insert into Fine table and update Status in Borrower table  IF v\_fine\_amt > 0 THEN  -- Insert the fine details into Fine table  INSERT INTO Fine (Roll\_no, Date, Amt)  VALUES (v\_roll\_no, SYSDATE, v\_fine\_amt);    -- Update the Status in Borrower table to 'R' (Returned)  UPDATE Borrower  SET Status = 'R'  WHERE Rollin = v\_roll\_no AND NameofBook = v\_name;    -- Commit the changes to database  COMMIT;  ELSE  -- If no fine, just update the status of the book to 'R'  UPDATE Borrower  SET Status = 'R'  WHERE Rollin = v\_roll\_no AND NameofBook = v\_name;    -- Commit the changes to database  COMMIT;  END IF;    -- Output the fine amount if any  IF v\_fine\_amt > 0 THEN  DBMS\_OUTPUT.PUT\_LINE('Fine to be paid: Rs. ' || v\_fine\_amt);  ELSE  DBMS\_OUTPUT.PUT\_LINE('No fine. Book returned on time.');  END IF;    EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('No record found for the provided Roll number and Book name.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);  END;  / |
|  |  |
| **M5** | **Design and Develop MongoDB Queries using Aggregation operations:**  Create Employee collection by considering following Fields: i. Emp\_id : Number   1. Name: Embedded Doc (FName, LName) 2. Company Name: String 3. Salary: Number 4. Designation: String 5. Age: Number 6. Expertise: Array 7. DOB: String or Date 8. Email id: String x. Contact: String   xi. Address: Array of Embedded Doc (PAddr, LAddr)  Insert at least 5 documents in collection by considering above attribute and execute following:   1. Using aggregation Return separates value in the Expertise array and return sum of each element of array.   db.Employee.aggregate([  { $unwind: "$Expertise" }, // Unwind the expertise array to separate each expertise into a document  { $group: { \_id: "$Expertise", totalCount: { $sum: 1 } } } // Group by expertise and count the occurrences  ])   1. Using Aggregate method return Max and Min Salary for each company.   db.Employee.aggregate([  { $group: {  \_id: "$Company Name",  maxSalary: { $max: "$Salary" },  minSalary: { $min: "$Salary" }  }}  ])   1. Using Aggregate method find Employee with Total Salary for Each City with Designation="DBA".   db.Employee.aggregate([  { $match: { "Designation": "DBA" } }, // Filter by designation "DBA"  { $group: {  \_id: { "city": { $arrayElemAt: ["$Address.LAddr", 0] } }, // Group by city  totalSalary: { $sum: "$Salary" }  }}  ])   1. Using aggregation method Return separates value in the Expertise array for employee name where Swapnil Jadhav   db.Employee.aggregate([  { $match: { "Name.FName": "Swapnil", "Name.LName": "Jadhav" } }, // Filter by employee name  { $unwind: "$Expertise" }, // Unwind the expertise array  { $project: { Expertise: 1 } } // Return expertise field  ])   1. To Create Compound Indexes on Name: 1, Age: -1   db.Employee.createIndex({ "Name.FName": 1, "Age": -1 })   1. Create an Index on Emp\_id field, compare the time require to search Emp\_id before and after creating an index. (Hint Add at least 10000 Documents)   // To test search time without index  let startTime = new Date();  db.Employee.find({ "Emp\_id": 5000 }).toArray(); // Search for a specific Emp\_id  let endTime = new Date();  print("Time without index:", endTime - startTime, "ms");  db.Employee.find({ "Emp\_id": 5000 }).explain("executionStats")  db.Employee.createIndex({ "Emp\_id": 1 })  db.Employee.find({ "Emp\_id": 5000 }).explain("executionStats")  // To test search time after index creation  startTime = new Date();  db.Employee.find({ "Emp\_id": 5000 }).toArray(); // Search for a specific Emp\_id  endTime = new Date();  print("Time with index:", endTime - startTime, "ms");   1. Return a List of Indexes on created on employee Collection.   db.Employee.getIndexes() |
|  |  |