|  |  |
| --- | --- |
| **1** | **SQL Queries**  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 TABLE Account (**  **Acc\_no INT PRIMARY KEY,**  **branch\_name VARCHAR(255),**  **balance DECIMAL(10, 2),**  **FOREIGN KEY (branch\_name) REFERENCES branch(branch\_name)**  **);**  **CREATE TABLE branch (**  **branch\_name VARCHAR(255) PRIMARY KEY,**  **branch\_city VARCHAR(255),**  **assets DECIMAL(15, 2) CHECK (assets >= 0)**  **);**  **CREATE TABLE customer (**  **cust\_name VARCHAR(255) PRIMARY KEY,**  **cust\_street VARCHAR(255),**  **cust\_city VARCHAR(255)**  **);**  **CREATE TABLE Depositor (**  **cust\_name VARCHAR(255),**  **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)**  **);**  **CREATE TABLE Loan (**  **loan\_no INT PRIMARY KEY,**  **branch\_name VARCHAR(255),**  **amount DECIMAL(15, 2) CHECK (amount >= 0),**  **FOREIGN KEY (branch\_name) REFERENCES branch(branch\_name)**  **);**  **CREATE TABLE Borrower (**  **cust\_name VARCHAR(255),**  **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)**  **);**  -- Insert data into the 'branch' table  INSERT INTO branch (branch\_name, branch\_city, assets) VALUES  ('Branch1', 'City1', 500000.00),  ('Branch2', 'City2', 750000.50),  ('Branch3', 'City3', 300000.75),  ('Branch4', 'City4', 1000000.25),  ('Branch5', 'City5', 600000.80);  -- Insert data into the 'customer' table  INSERT INTO customer (cust\_name, cust\_street, cust\_city) VALUES  ('Customer1', 'Street1', 'City1'),  ('Customer2', 'Street2', 'City2'),  ('Customer3', 'Street3', 'City3'),  ('Customer4', 'Street4', 'City4'),  ('Customer5', 'Street5', 'City5');  -- Insert data into the 'Account' table  INSERT INTO Account (Acc\_no, branch\_name, balance) VALUES  (1, 'Branch1', 1500.00),  (2, 'Branch2', 2000.50),  (3, 'Branch3', 500.75),  (4, 'Branch4', 10000.25),  (5, 'Branch5', 800.80);  -- Insert data into the 'Depositor' table  INSERT INTO Depositor (cust\_name, acc\_no) VALUES  ('Customer1', 1),  ('Customer2', 2),  ('Customer3', 3),  ('Customer4', 4),  ('Customer5', 5);  -- Insert data into the 'Loan' table  INSERT INTO Loan (loan\_no, branch\_name, amount) VALUES  (101, 'Branch1', 5000.00),  (102, 'Branch2', 10000.50),  (103, 'Branch3', 7500.75),  (104, 'Branch4', 20000.25),  (105, 'Branch5', 12000.80);  -- Insert data into the 'Borrower' table  INSERT INTO Borrower (cust\_name, loan\_no) VALUES  ('Customer1', 101),  ('Customer2', 102),  ('Customer3', 103),  ('Customer4', 104),  ('Customer5', 105);  -- Set column formatting for the 'borrower' table  COLUMN cust\_name FORMAT A20  COLUMN loan\_no FORMAT 9999  -- Display data from the 'borrower' table  SELECT \* FROM borrower;  -- Set column formatting for the 'loan' table  COLUMN loan\_no FORMAT 9999  COLUMN branch\_name FORMAT A20  COLUMN amount FORMAT 99999.99  -- Display data from the 'loan' table  SELECT \* FROM loan;  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)  Solve following query:  Create above tables with appropriate constraints like primary key, foreign key, check constraints, not null etc.   1. Find the names of all branches in loan relation. 2. Find all loan numbers for loans made at 'Shivaji Nagar' Branch with loan amount > 12000. 3. Find all customers who have a loan from bank. Find their names,loan\_no and loan amount. 4. List all customers in alphabetical order who have loan from Akurdi branch.   **-- Solve the queries**  **-- a) Find the names of all branches in the loan relation.**  **SELECT DISTINCT branch\_name FROM Loan;**  **-- b) Find all loan numbers for loans made at 'Shivaji Nagar' Branch with loan amount > 12000.**  **SELECT loan\_no FROM Loan WHERE branch\_name = 'Shivaji Nagar' AND amount > 12000;**  **-- c) Find all customers who have a loan from the bank. Find their names, loan\_no, and loan amount.**  **SELECT c.cust\_name, l.loan\_no, l.amount**  **FROM customer c**  **JOIN Borrower b ON c.cust\_name = b.cust\_name**  **JOIN Loan l ON b.loan\_no = l.loan\_no;**  **-- d) List all customers in alphabetical order who have a loan from the Akurdi branch.**  **SELECT c.cust\_name**  **FROM customer c**  **JOIN Borrower b ON c.cust\_name = b.cust\_name**  **JOIN Loan l ON b.loan\_no = l.loan\_no**  **WHERE l.branch\_name = 'Akurdi'**  **ORDER BY c.cust\_name;** |
| **2** | **SQL Queries**  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)  Solve following query:   1. Find all customers who have an account or loan or both at bank. 2. Find all customers who have both account and loan at bank. 3. Find all customers who have account but no loan at the bank. 4. Find average account balance at Akurdi branch.   SELECT DISTINCT c.cust\_name  FROM customer c  LEFT JOIN depositor d ON c.cust\_name = d.cust\_name  LEFT JOIN borrower b ON c.cust\_name = b.cust\_name  WHERE d.cust\_name IS NOT NULL OR b.cust\_name IS NOT NULL;  SELECT DISTINCT c.cust\_name  FROM customer c  JOIN depositor d ON c.cust\_name = d.cust\_name  JOIN borrower b ON c.cust\_name = b.cust\_name;  SELECT DISTINCT c.cust\_name  FROM customer c  JOIN depositor d ON c.cust\_name = d.cust\_name  LEFT JOIN borrower b ON c.cust\_name = b.cust\_name  WHERE b.cust\_name IS NULL;  SELECT AVG(balance) AS avg\_balance  FROM account  WHERE branch\_name = 'Akurdi'; |
| **3** | **SQL Queries**  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)  Solve following query:   1. Find the branches where average account balance > 15000. 2. Find number of tuples in customer relation. 3. Calculate total loan amount given by bank. 4. Delete all loans with loan amount between 1300 and 1500.   -- a) Find the branches where average account balance > 15000.  SELECT branch\_name  FROM Account  GROUP BY branch\_name  HAVING AVG(balance) > 15000;  -- b) Find the number of tuples in the customer relation.  SELECT COUNT(\*)  FROM customer;  -- c) Calculate the total loan amount given by the bank.  SELECT SUM(amount) AS total\_loan\_amount  FROM Loan;  -- d) Delete all loans with a loan amount between 1300 and 1500.  DELETE FROM Loan  WHERE amount BETWEEN 1300 AND 1500; |
| **4** | **SQL Queries**  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)  C***reate Table, View, Index, Sequence, Synonym ,different constraints for above schema***  -- Create tables  CREATE TABLE Account (  Acc\_no INT PRIMARY KEY,  branch\_name VARCHAR(255) NOT NULL,  balance DECIMAL(10, 2),  CONSTRAINT fk\_branch\_acc FOREIGN KEY (branch\_name) REFERENCES branch(branch\_name)  );  CREATE TABLE branch (  branch\_name VARCHAR(255) PRIMARY KEY,  branch\_city VARCHAR(255) NOT NULL,  assets DECIMAL(15, 2) CHECK (assets >= 0)  );  CREATE TABLE customer (  cust\_name VARCHAR(255) PRIMARY KEY,  cust\_street VARCHAR(255),  cust\_city VARCHAR(255)  );  CREATE TABLE Depositor (  cust\_name VARCHAR(255),  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)  );  CREATE TABLE Loan (  loan\_no INT PRIMARY KEY,  branch\_name VARCHAR(255) NOT NULL,  amount DECIMAL(15, 2) CHECK (amount >= 0),  CONSTRAINT fk\_loan\_branch FOREIGN KEY (branch\_name) REFERENCES branch(branch\_name)  );  CREATE TABLE Borrower (  cust\_name VARCHAR(255),  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 a view  CREATE VIEW CustomerAccountView AS  SELECT c.cust\_name, c.cust\_city, a.Acc\_no, a.balance  FROM customer c  JOIN Depositor d ON c.cust\_name = d.cust\_name  JOIN Account a ON d.acc\_no = a.Acc\_no;  -- Create an index  CREATE INDEX idx\_loan\_amount ON Loan(amount);  -- Create a sequence  CREATE SEQUENCE loan\_seq START WITH 1 INCREMENT BY 1;  -- Create a synonym  CREATE SYNONYM cust\_synonym FOR customer; |
| **5** | **SQL Queries**  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)  **all types of Join, Sub-Query and View**  -- Create tables  CREATE TABLE Account (  Acc\_no INT PRIMARY KEY,  branch\_name VARCHAR(255) NOT NULL,  balance DECIMAL(10, 2),  CONSTRAINT fk\_branch\_acc FOREIGN KEY (branch\_name) REFERENCES branch(branch\_name)  );  CREATE TABLE branch (  branch\_name VARCHAR(255) PRIMARY KEY,  branch\_city VARCHAR(255) NOT NULL,  assets DECIMAL(15, 2) CHECK (assets >= 0)  );  CREATE TABLE customer (  cust\_name VARCHAR(255) PRIMARY KEY,  cust\_street VARCHAR(255),  cust\_city VARCHAR(255)  );  CREATE TABLE Depositor (  cust\_name VARCHAR(255),  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)  );  CREATE TABLE Loan (  loan\_no INT PRIMARY KEY,  branch\_name VARCHAR(255) NOT NULL,  amount DECIMAL(15, 2) CHECK (amount >= 0),  CONSTRAINT fk\_loan\_branch FOREIGN KEY (branch\_name) REFERENCES branch(branch\_name)  );  CREATE TABLE Borrower (  cust\_name VARCHAR(255),  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)  );  -- Insert sample data into the tables (adjust values as needed)  -- Perform different types of joins  -- 1. INNER JOIN  SELECT \*  FROM customer c  JOIN Depositor d ON c.cust\_name = d.cust\_name;  -- 2. LEFT JOIN  SELECT \*  FROM branch b  LEFT JOIN Account a ON b.branch\_name = a.branch\_name;  -- 3. RIGHT JOIN  SELECT \*  FROM Loan l  RIGHT JOIN Borrower b ON l.loan\_no = b.loan\_no;  -- 4. FULL OUTER JOIN  SELECT \*  FROM Depositor d  FULL OUTER JOIN Account a ON d.acc\_no = a.Acc\_no;  -- Perform subqueries  -- 1. Simple subquery  SELECT \*  FROM customer  WHERE cust\_name IN (SELECT cust\_name FROM Depositor);  -- 2. Correlated subquery  SELECT \*  FROM Loan l  WHERE amount > (SELECT AVG(amount) FROM Loan WHERE l.branch\_name = branch\_name);  -- Create a view  CREATE VIEW CustomerAccountView AS  SELECT c.cust\_name, c.cust\_city, a.Acc\_no, a.balance  FROM customer c  JOIN Depositor d ON c.cust\_name = d.cust\_name  JOIN Account a ON d.acc\_no = a.Acc\_no; |
| **6** | **SQL queries**  Create table Cust\_Master(Cust\_no, Cust\_name, Qty\_Ordered, Order\_date, Cust\_addr )  Cust\_no is defined as primary key,  Insert ten records in the table.   * List names of customers having 'a' as second letter in their name. * List customers who stay in city whose first letter is 'M' * Display order from Customer no C1002,C1005,C1007 and C1008 * List Clients who stay in either 'Banglore or 'Manglore' * Create view Customer\_View consisting of Cust\_no, Qty\_ordered and Order\_date   -- Create Cust\_Master Table  CREATE TABLE Cust\_Master (  Cust\_no INT PRIMARY KEY,  Cust\_name VARCHAR(255),  Qty\_Ordered INT,  Order\_date DATE,  Cust\_addr VARCHAR(255)  );  -- Insert 10 records  INSERT INTO Cust\_Master (Cust\_no, Cust\_name, Qty\_Ordered, Order\_date, Cust\_addr) VALUES  (1001, 'Alice', 5, '2023-01-15', 'Bangalore'),  (1002, 'Bob', 8, '2023-02-20', 'Mangalore'),  (1003, 'Charlie', 10, '2023-03-10', 'Mumbai'),  (1004, 'David', 7, '2023-04-05', 'Chennai'),  (1005, 'Eva', 12, '2023-05-12', 'Mysore'),  (1006, 'Frank', 3, '2023-06-18', 'Delhi'),  (1007, 'Grace', 6, '2023-07-25', 'Pune'),  (1008, 'Henry', 9, '2023-08-30', 'Mangalore'),  (1009, 'Ivy', 4, '2023-09-07', 'Bangalore'),  (1010, 'Jack', 11, '2023-10-15', 'Hyderabad');  -- List names of customers having 'a' as the second letter in their name  SELECT Cust\_name  FROM Cust\_Master  WHERE SUBSTR(Cust\_name, 2, 1) = 'a';  -- List customers who stay in a city whose first letter is 'M'  SELECT \*  FROM Cust\_Master  WHERE SUBSTR(Cust\_addr, 1, 1) = 'M';  -- Display orders from Customer no C1002, C1005, C1007, and C1008  SELECT \*  FROM Cust\_Master  WHERE Cust\_no IN (1002, 1005, 1007, 1008);  -- List clients who stay in either 'Bangalore' or 'Mangalore'  SELECT \*  FROM Cust\_Master  WHERE Cust\_addr IN ('Bangalore', 'Mangalore');  -- Create view Customer\_View consisting of Cust\_no, Qty\_ordered, and Order\_date  CREATE VIEW Customer\_View AS  SELECT Cust\_no, Qty\_Ordered, Order\_date  FROM Cust\_Master; |
| **7** | **SQL queries**  Create following tables with suitable constraints.  Insert data and solve the following queries:  PROPERTIES(Pno, Type, Sq\_Ft\_Area, Rent, Address, Status, Owner No)  OWNERS (OwnerNo, OwnerName, Phno)   1. Display all 1 BHK apartments in Kothrud which are not rented 2. Display all properties owned by “Gopal” 3. Write a query to display the smallest property of each owner 4. Display all properties in Kothrud in Descending order of rent   -- Creating PROPERTIES table  CREATE TABLE PROPERTIES (  Pno INT PRIMARY KEY,  Type VARCHAR(50),  Sq\_Ft\_Area INT,  Rent INT,  Address VARCHAR(100),  Status VARCHAR(50),  OwnerNo INT,  FOREIGN KEY (OwnerNo) REFERENCES OWNERS(OwnerNo)  );  -- Creating OWNERS table  CREATE TABLE OWNERS (  OwnerNo INT PRIMARY KEY,  OwnerName VARCHAR(100),  Phno VARCHAR(15)  );  -- Inserting sample data into OWNERS table  INSERT INTO OWNERS VALUES (1, 'Gopal', '1234567890');  INSERT INTO OWNERS VALUES (2, 'AnotherOwner', '9876543210');  -- Inserting sample data into PROPERTIES table  INSERT INTO PROPERTIES VALUES (101, '1 BHK', 800, 12000, 'Kothrud', 'Not Rented', 1);  INSERT INTO PROPERTIES VALUES (102, '2 BHK', 1200, 20000, 'Baner', 'Rented', 1);  INSERT INTO PROPERTIES VALUES (103, '1 BHK', 700, 10000, 'Kothrud', 'Not Rented', 2);  INSERT INTO PROPERTIES VALUES (104, 'Studio', 500, 8000, 'Kothrud', 'Rented', 2);  -- 1. Display all 1 BHK apartments in Kothrud which are not rented  SELECT \*  FROM PROPERTIES  WHERE Type = '1 BHK' AND Address = 'Kothrud' AND Status = 'Not Rented';  -- 2. Display all properties owned by “Gopal”  SELECT \*  FROM PROPERTIES  WHERE OwnerNo = (SELECT OwnerNo FROM OWNERS WHERE OwnerName = 'Gopal');  -- 3. Write a query to display the smallest property of each owner  SELECT \*  FROM PROPERTIES P1  WHERE Sq\_Ft\_Area = (SELECT MIN(Sq\_Ft\_Area) FROM PROPERTIES P2 WHERE P1.OwnerNo = P2.OwnerNo);  -- 4. Display all properties in Kothrud in Descending order of rent  SELECT \*  FROM PROPERTIES  WHERE Address = 'Kothrud'  ORDER BY Rent DESC; |
| **8** | **SQL queries**  **Basic SQL** Create following tables with suitable constraints.  Insert data and solve the following queries:  PROPERTIES(Pno, Type, Sq\_Ft\_Area, Rent, Address, Status, Owner No)  OWNERS (OwnerNo, OwnerName, Phno)   1. Create a view which shows OwnerName along with his Pno, type, Address and Rent   CREATE VIEW OwnerPropertiesView AS  SELECT O.OwnerName, P.Pno, P.Type, P.Address, P.Rent  FROM OWNERS O  JOIN PROPERTIES P ON O.OwnerNo = P.OwnerNo;   1. Display the names of all Owners whose name has “ee” 2. Display Pno, Type, Address, Rent and status of all properties with rent greater than 15000/- and less than 22000/- 3. Display different property types registered with the real estate agency   select distinct type from properties; |
| **9** | **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)   1. List all stationary items with price between 400/- to 1000/- 2. Change the mobile number of customer “Gopal”   -- Update the PURCHASE table first  UPDATE PURCHASE  SET Cno = 7821836970  WHERE Cno = (SELECT CNo FROM CUSTOMERS WHERE Cname = 'Gopal');  -- Now, update the CUSTOMERS table  UPDATE CUSTOMERS  SET Cno = 7821836970  WHERE Cname = 'Gopal';  While there are integrity constrain as foreign key  se   1. Display the item with maximum price 2. Display all purchases sorted from the most recent to the oldest   -- Create CUSTOMERS table  CREATE TABLE CUSTOMERS (  CNo INT PRIMARY KEY,  Cname VARCHAR(100),  Ccity VARCHAR(50),  CMobile VARCHAR(15)  );  -- Create ITEMS table  CREATE TABLE ITEMS (  INo INT PRIMARY KEY,  Iname VARCHAR(100),  Itype VARCHAR(50),  Iprice INT,  Icount INT  );  -- Create PURCHASE table  CREATE TABLE PURCHASE (  PNo INT PRIMARY KEY,  Pdate DATE,  Pquantity INT,  Cno INT,  INo INT,  FOREIGN KEY (Cno) REFERENCES CUSTOMERS(CNo),  FOREIGN KEY (INo) REFERENCES ITEMS(INo)  );  -- Insert sample data into CUSTOMERS table  INSERT INTO CUSTOMERS VALUES (1, 'Gopal', 'Pune', '9876543210');  INSERT INTO CUSTOMERS VALUES (2, 'Alice', 'New York', '1234567890');  -- Insert sample data into ITEMS table  INSERT INTO ITEMS VALUES (101, 'Pen', 'Stationary', 10, 500);  INSERT INTO ITEMS VALUES (102, 'Notebook', 'Stationary', 50, 200);  INSERT INTO ITEMS VALUES (103, 'Mobile Phone', 'Electronics', 15000, 10);  -- Insert sample data into PURCHASE table  INSERT INTO PURCHASE VALUES (201, '2023-01-15', 2, 1, 101);  INSERT INTO PURCHASE VALUES (202, '2023-02-10', 1, 2, 102);  INSERT INTO PURCHASE VALUES (203, '2023-03-05', 3, 1, 103);  -- 1. List all stationary items with price between 400/- to 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 = '9999999999'  WHERE Cname = 'Gopal';  -- 3. Display the item with the maximum price  SELECT \*  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; |
| **10** | **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)   1. Count the number of customers in every city 2. Display all purchased quantity of Customer Maya 3. Display list of customers whose name ends with 'a' 4. Create view which shows Iname, Price and Count of all stationary items in descending order of price   -- Create CUSTOMERS table  CREATE TABLE CUSTOMERS (  CNo INT PRIMARY KEY,  Cname VARCHAR(100),  Ccity VARCHAR(50),  CMobile VARCHAR(15)  );  -- Create ITEMS table  CREATE TABLE ITEMS (  INo INT PRIMARY KEY,  Iname VARCHAR(100),  Itype VARCHAR(50),  Iprice INT,  Icount INT  );  -- Create PURCHASE table  CREATE TABLE PURCHASE (  PNo INT PRIMARY KEY,  Pdate DATE,  Pquantity INT,  Cno INT,  INo INT,  FOREIGN KEY (Cno) REFERENCES CUSTOMERS(CNo),  FOREIGN KEY (INo) REFERENCES ITEMS(INo)  );  -- Insert sample data into CUSTOMERS table  INSERT INTO CUSTOMERS VALUES (1, 'Maya', 'New York', '9876543210');  INSERT INTO CUSTOMERS VALUES (2, 'Alice', 'London', '1234567890');  INSERT INTO CUSTOMERS VALUES (3, 'Aisha', 'New York', '8765432109');  -- Insert sample data into ITEMS table  INSERT INTO ITEMS VALUES (101, 'Pen', 'Stationary', 10, 500);  INSERT INTO ITEMS VALUES (102, 'Notebook', 'Stationary', 50, 200);  INSERT INTO ITEMS VALUES (103, 'Mobile Phone', 'Electronics', 15000, 10);  -- Insert sample data into PURCHASE table  INSERT INTO PURCHASE VALUES (201, '2023-01-15', 2, 1, 101);  INSERT INTO PURCHASE VALUES (202, '2023-02-10', 1, 2, 102);  INSERT INTO PURCHASE VALUES (203, '2023-03-05', 3, 1, 103);  -- 1. Count the number of customers in every city  SELECT Ccity, COUNT(\*) AS CustomerCount  FROM CUSTOMERS  GROUP BY Ccity;  -- 2. Display all purchased quantity of Customer Maya  SELECT Pquantity  FROM PURCHASE  WHERE Cno = (SELECT CNo FROM CUSTOMERS WHERE Cname = 'Maya');  -- 3. Display list of customers whose name ends with 'a'  SELECT \*  FROM CUSTOMERS  WHERE Cname LIKE '%a';  -- 4. Create a view which shows Iname, Price, and Count of all stationary items in descending order of price  CREATE VIEW StationaryItemsView AS  SELECT Iname, Iprice, Icount  FROM ITEMS  WHERE Itype = 'Stationary'  ORDER BY Iprice DESC; |
| **11** | **PL/SQL**  Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 5 to   1. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area   SQL> CREATE TABLE circle(radius NUMBER, area NUMBER);  Table created.  SQL> DECLARE  2 radius\_var NUMBER;  3 area\_var NUMBER;  4 pi NUMBER := 3.14;  5 BEGIN  6 FOR radius\_var IN 5 .. 9 LOOP  7 area\_var := pi\*radius\_var\*radius\_var;  8 dbms\_output.put\_line(area\_var);  9 INSERT INTO circle VALUES (radius\_var,area\_var);  10 END LOOP;  11 END;  12 /  PL/SQL procedure successfully completed.  SQL> select \* from circle; |
| **12** | **PL/SQL procedure**   1. Borrower(Roll\_no, Name, DateofIssue, NameofBook, Status) 2. Fine(Roll\_no,Date,Amt)   Accept Roll\_no and NameofBook from user.  Write a PL/SQL procedure. The procedure calculates the fine and performs the book returning operation  Assume suitable conditions for calculating fine.   * + Check the number of days (from date of issue).   + If days are between 15 to 30 then fine amount will be Rs 5per day.   + If no. of days>30, per day fine will be Rs 50 per day and for days less than 30, Rs. 5 per day.   + If condition of fine is true, then details will be stored into fine table.   + Also handles the exception by named exception handler or user define exception handler.   create table Borrower (RollNo NUMBER PRIMARY KEY,Name varchar(10),IssueDate Date,Status varchar(10),BookName varchar(10));  create table Fine (RollNo NUMBER,ReturnDate Date,Amount NUMBER);  fill info in just borrower table  CREATE OR REPLACE PROCEDURE ReturnBook (  p\_RollNo IN NUMBER,  p\_BookName IN VARCHAR2  ) IS  v\_DateofIssue DATE;  v\_CurrentDate DATE := SYSDATE;  v\_NumDays NUMBER;  v\_FineAmount NUMBER := 0;  BEGIN  -- Retrieve DateofIssue for the given RollNo and BookName  SELECT IssueDate INTO v\_DateofIssue  FROM Borrower  WHERE RollNo = p\_RollNo  AND BookName = p\_BookName;  -- Calculate the number of days between DateofIssue and CurrentDate  v\_NumDays := TRUNC(v\_CurrentDate - v\_DateofIssue);  -- Check conditions for fine calculation  IF v\_NumDays BETWEEN 15 AND 30 THEN  v\_FineAmount := v\_NumDays \* 5; -- Rs 5 per day  ELSIF v\_NumDays > 30 THEN  v\_FineAmount := (v\_NumDays - 30) \* 50 + 150; -- Rs 50 per day after 30 days  END IF;  -- If a fine is applicable, store details in the Fine table  IF v\_FineAmount > 0 THEN  INSERT INTO Fine (RollNo, ReturnDate, Amount)  VALUES (p\_RollNo, v\_CurrentDate, v\_FineAmount);  END IF;  -- Update the Status in Borrower table to 'Returned'  UPDATE Borrower  SET Status = 'Returned'  WHERE RollNo = p\_RollNo  AND BookName = p\_BookName;  COMMIT;  -- Display success message  DBMS\_OUTPUT.PUT\_LINE('Book returned successfully.');  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  -- Handle the case where no data is found for the given RollNo and BookName  DBMS\_OUTPUT.PUT\_LINE('Book not found for the given RollNo and BookName.');  WHEN OTHERS THEN  -- Handle other exceptions and display the error message  DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);  END ReturnBook;  /  SET SERVEROUTPUT ON;  -- Call the procedure and provide values for RollNo and BookName  EXEC ReturnBook(p\_RollNo, 'BookTitle'); |
| **13** | **Procedure**  Consider table Student (Roll, Name, Attendance ,Status)  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”  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  create table Student(RollNo NUMBER PRIMARY KEY,Name varchar(10),Attendance NUMBER,Status varchar(10) DEFAULT 'NOT SET');  INSERT INTO Student (RollNo, Name, Attendance) VALUES (1, 'Pruthvi', 90);  INSERT INTO Student (RollNo, Name, Attendance) VALUES (2, 'Adi', 75);  INSERT INTO Student (RollNo, Name, Attendance) VALUES (3, 'Varun', 14);  INSERT INTO Student (RollNo, Name, Attendance) VALUES (4, 'Prajakta', 58);  INSERT INTO Student (RollNo, Name, Attendance) VALUES (5, 'Nayan', 36);  INSERT INTO Student (RollNo, Name, Attendance) VALUES (6, 'Samarth', 88);  INSERT INTO Student (RollNo, Name, Attendance) VALUES (7, 'Dhanu', 78);  CREATE OR REPLACE PROCEDURE CheckAttendanceStatus (  p\_RollNo IN NUMBER  )  IS  vAttendance NUMBER;  BEGIN  SELECT Attendance INTO vAttendance FROM Student WHERE RollNo = p\_RollNo;  IF vAttendance < 75 THEN  dbms\_output.put\_line('Term Not Granted');  UPDATE Student SET Status = 'Detained' WHERE RollNo = p\_RollNo;  ELSE  dbms\_output.put\_line('Term Granted');  UPDATE Student SET Status = 'Not Detain' WHERE RollNo = p\_RollNo;  END IF;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  dbms\_output.put\_line('No Such Data Found');  WHEN OTHERS THEN  dbms\_output.put\_line('Internal SQL Error');  END CheckAttendanceStatus;  /  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| **14** | **Procedure**  **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 category 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(Roll, Name, Total\_marks)  Result(Roll, Name, Class)  create table Student (RollNo NUMBER PRIMARY KEY,Name varchar(10),Marks NUMBER);  insert into Student values (1,'Pruthvi',900);  insert into Student values (2,'Dhanu',900);  insert into Student values (3,'Adi',1400);  insert into Student values (4,'Nayan',850);  insert into Student values (5,'Samarth',950);  create table Result(RollNo NUMBER,Name varchar(10),Class varchar(14));  CREATE OR REPLACE PROCEDURE Proc\_Grade (  pMarks IN NUMBER,  pClass OUT varchar  ) AS  BEGIN  IF pMarks >= 990 AND pMarks <= 1500 THEN  pClass := 'Destinction';  ELSIF pMarks >= 900 AND pMarks <= 989 THEN  pClass := 'First Class';  ELSIF pMarks >= 825 AND pMarks <= 899 THEN  pClass := 'Second Class';  ELSE  pClass := 'Fail';  END IF;  END;  /  DECLARE  vRollNo NUMBER;  vMarks NUMBER;  vName varchar(10);  vClass varchar(14);  BEGIN  vRollNo := &rollno;  SELECT Marks INTO vMarks FROM Student WHERE RollNo=vRollNo;  SELECT Name INTO vName FROM Student WHERE RollNo=vRollNo;  Proc\_Grade(vMarks,vClass);  INSERT INTO RESULT VALUES (vRollNo,vName,vClass);  COMMIT;  END;  / |
| **15** | **Cursor**  Write a PL/SQL block of code using Cursor that will merge the data available in the newly created table N\_RollCall with the data available in the table O\_RollCall. If the data in the first table already exist in the second table then that data should be skipped.  create table new\_class(roll number(10), name varchar(10));  create table old\_class(roll number(10), name varchar(10));  insert into new\_class values(401 ,'Aditi');  insert into new\_class values(402 ,'Saumya');  insert into old\_class values(401 ,'Atmika');  insert into old\_class values(402 ,'Mohit');  insert into old\_class values(403 ,'Nitin');  insert into old\_class values(404 ,'Seema');  select \* from old\_class;  select \* from new \_class;  declare  cursor cur\_class is select \* from old\_class;  cursor cur\_check(str\_name varchar) is select roll from new\_class where name =str\_name;  str\_roll new\_class.roll%type;  str\_name new\_class.name%type;  load number(10);  Begin  open cur\_class;  loop  fetch cur\_class into str\_roll,str\_name;  Exit when cur\_class%notfound;  open cur\_check(str\_name);  fetch cur\_check into load;  if cur\_check%found then  dbms\_output.put\_line('stud' || ' ' || str\_name || ' ' || 'Name is already there .' );  else  dbms\_output.put\_line('stud' || ' ' || str\_name || ' ' || 'Name does not exist . Inserting in new\_class table' );  insert into new\_class values(str\_roll,str\_name);  end if;  close cur\_check;  end loop;  close cur\_class;  end; |
| **16** | **Trigger**  Write a after trigger for Insert, update and delete event considering following requirement:  Emp(Emp\_no, Emp\_name, Emp\_salary)   1. Trigger should be initiated when salary tried to be inserted is less than Rs.50,000/- 2. Trigger should be initiated when salary tried to be updated for value less than Rs. 50,000/- 3. Also the new values expected to be inserted will be stored in new table Tracking(Emp\_no,Emp\_salary).   create table Emp(EmpNo NUMBER PRIMARY KEY,EmpName varchar(10),EmpSalary NUMBER);  create table Tracking(EmpNo NUMBER,EmpSalary NUMBER,Action varchar(10));  CREATE OR REPLACE TRIGGER EmpTrigger  BEFORE INSERT OR UPDATE OR DELETE ON Emp  FOR EACH ROW  BEGIN  IF INSERTING THEN  IF :NEW.EmpSalary < 50000 THEN  INSERT INTO Tracking (EmpNo, EmpSalary, Action) values (:NEW.EmpNo,:NEW.EmpSalary,'Insert');  END IF;  ELSIF UPDATING THEN  IF :NEW.EmpSalary < 50000 THEN  INSERT INTO Tracking (EmpNo, EmpSalary, Action) values (:NEW.EmpNo,:NEW.EmpSalary,'Update');  END IF;  ELSE  INSERT INTO Tracking (EmpNo, EmpSalary, Action) values (:OLD.EmpNo,:OLD.EmpSalary,'Delete');  END IF;  END;  /  insert into Emp values (1,'Pruthvi',60000);  --update  --delete |
| **17** | **Trigger**  Consider CUSTOMER (ID, Name, Age, Address, Salary) create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.  CREATE TABLE CUST (  ID INT PRIMARY KEY,  Name VARCHAR(50),  Age INT,  Address VARCHAR(100),  Salary INT  );  INSERT INTO CUST VALUES (1, 'John', 25, '123 Main St', 50000);  INSERT INTO CUST VALUES (2, 'Jane', 30, '456 Oak St', 60000);  SET SERVEROUTPUT ON;  CREATE OR REPLACE TRIGGER SalaryDifferenceTrigger  BEFORE INSERT OR UPDATE OR DELETE ON CUST  FOR EACH ROW  DECLARE  OldSalary INT;  NewSalary INT;  BEGIN  IF INSERTING OR UPDATING THEN  OldSalary := NVL(:OLD.Salary, 0);  NewSalary := NVL(:NEW.Salary, 0);  DBMS\_OUTPUT.PUT\_LINE('Salary Difference: ' || (NewSalary - OldSalary));  ELSIF DELETING THEN  DBMS\_OUTPUT.PUT\_LINE('Deleted Salary: ' || :OLD.Salary);  END IF;  END;  /  -- Query the CUST table  SELECT \* FROM CUST;  -- Perform an UPDATE operation  UPDATE CUST SET Salary = 55000 WHERE ID = 1;  -- Perform a DELETE operation  DELETE FROM CUST WHERE ID = 2;  -- Perform an INSERT operation  INSERT INTO CUST VALUES (3, 'Alice', 28, '789 Elm St', 70000); |
| **18** | **MongoDB CRUD** operations,  Create collection Employee (Emp\_id, Emp\_Name, Emp\_salary, Emp\_Dept.)  Insert 10 Documents in the collection.  Find the employees whose salary is greater than 50000 Rs.  Increase the salary of Smith by 5000 Rs  Display the information of employees working in Marketing department.with less than 45000 salary .  Display first five highest paid employees  Delete Employee with Id ‘E1007’  Create an Index on Emp\_Id field , compare the time require to search Emp\_id ‘E10008’ before and after creating an index. (Hint Add at least 10000 Documents)  for (let i = 1; i <= 10000; i++) {  db.Employee.insertOne({  Emp\_Id: "E" + i,  Emp\_Name: "Employee " + i,  Emp\_salary: Math.floor(Math.random() \* 50000) + 50000, // Random salary between 50000 and 100000  Emp\_Dept: "IT"  });  }  let startTime = new Date();  db.Employee.find({ Emp\_Id: 'E10008' }).explain("executionStats");  let endTime = new Date();  let timeBeforeIndex = endTime - startTime;  print("Time Before Index: " + timeBeforeIndex + "ms");  db.Employee.createIndex({ Emp\_Id: 1 });  let startTime = new Date();  db.Employee.find({ Emp\_Id: 'E10008' }).explain("executionStats");  let endTime = new Date();  let timeAfterIndex = endTime - startTime;  print("Time After Index: " + timeAfterIndex + "ms"); |
| **19** | **MapReduce**  Create a customer collection consisting of fields like name, email ID, profession, gender, bill amount   1. Write a MapReduce query for finding the count of male and female customers in the collection   var mapFunction1 = function() {  emit(this.gender, 1);  };  var reduceFunction1 = function(key, values) {  return Array.sum(values);  };  db.customer.mapReduce(  mapFunction1,  reduceFunction1,  { out: "gender\_count" }  );  db.gender\_count.find();   1. Write a MapReduce query for finding the count of each profession in the collection   var mapFunction2 = function() {  emit(this.profession, 1);  };  var reduceFunction2 = function(key, values) {  return Array.sum(values);  };  db.customer.mapReduce(  mapFunction2,  reduceFunction2,  { out: "profession\_count" }  );  db.profession\_count.find();   1. Display list of all customers with bill amounts greater than 5000/- 2. Update the bill amount of any one customer 3. Display all customers with name starting with 'B' 4. Display list of all customers with profession = “Business” 5. Display all customers in Descending order of Bill amount 6. Create an index on name field of customer collection. Also use the explain() function |
| **20** | **Mongo Aggregation**  Create a student collection consisting of fields like Roll No, name, class, marks, sports etc   1. Display the first 5 toppers of TE 2. Display marks of topper of each division 3. Display the average marks of each division 4. Display the rollcall of TE Comp A 5. Display list of fail students from TE Comp A 6. Display Name, Class and Marks of all students 7. Display list of students who play football 8. Display the first 5 toppers of TE   db.students.aggregate([  { $match: { class: "TE" } },  { $sort: { marks: -1 } },  { $limit: 5 },  { $project: { \_id: 0, RollNo: 1, name: 1, class: 1, marks: 1 } }  ])   1. Display marks of topper of each division   db.students.aggregate([  {  $group: {  \_id: { class: "$class", division: "$division" },  topper: { $first: "$$ROOT" },  maxMarks: { $max: "$marks" }  }  },  {  $project: {  \_id: 0,  class: "$\_id.class",  division: "$\_id.division",  topperName: "$topper.name",  topperMarks: "$maxMarks"  }  }  ])   1. Display the average marks of each division   db.students.aggregate([  { $group: { \_id: { class: "$class", division: "$division" }, avgMarks: { $avg: "$marks" } } },  { $project: { \_id: 0, class: "$\_id.class", division: "$\_id.division", avgMarks: 1 } }  ])   1. Display the rollcall of TE Comp A   db.students.find({ class: "TE", division: "Comp A" }, { \_id: 0, RollNo: 1, name: 1 })   1. Display list of fail students from TE Comp A   db.students.find({ class: "TE", division: "Comp A", marks: { $lt: 40 } }, { \_id: 0, RollNo: 1, name: 1, marks: 1 })   1. Display Name, Class and Marks of all students   db.students.find({}, { \_id: 0, name: 1, class: 1, marks: 1 })   1. Display list of students who play football   db.students.find({ sports: "football" }, { \_id: 0, name: 1, class: 1, marks: 1, sports: 1 }) |