|  |  |
| --- | --- |
| **Sr No** | **Assignment statement** |
| 01 | To study DDL-create and DML-insert commands .  (i)Create tables according to the following definition. CREATE TABLE DEPOSIT (ACTNO VARCHAR2(5), CNAME VARCHAR2(18), BNAME VARCHAR2(18), AMOUNT NUMBER(8,2) , ADATE DATE);  CREATE TABLE BRANCH(BNAME VARCHAR2(18),CITY VARCHAR2(18));  CREATE TABLE CUSTOMERS(CNAME VARCHAR2(19) ,CITY VARCHAR2(18));  CREATE TABLE BORROW(LOANNO VARCHAR2(5), CNAME VARCHAR2(18), BNAME VARCHAR2(18), AMOUNT NUMBER (8,2));  (ii) Insert the data for all tables.  From the above given tables perform the following queries:  (1) Describe deposit, branch.  (2) List all data from table DEPOSIT  (3) List all data from table BORROW.  (4) Give account number and amount of depositors.  (5) Give name of depositors having amount greater than 4000.  (6) Give name of customers who opened account after date ‘1-12-96’ |
|  | Create Database Deposit;  use Deposit;  -- Create DEPOSIT table  CREATE TABLE DEPOSIT (  ACTNO VARCHAR(5),  CNAME VARCHAR(18),  BNAME VARCHAR(18),  AMOUNT DECIMAL(8,2),  ADATE DATE  );  -- Create BRANCH table  CREATE TABLE BRANCH (  BNAME VARCHAR(18),  CITY VARCHAR(18)  );  -- Create CUSTOMERS table  CREATE TABLE CUSTOMERS (  CNAME VARCHAR(19),  CITY VARCHAR(18)  );  -- Create BORROW table  CREATE TABLE BORROW (  LOANNO VARCHAR(5),  CNAME VARCHAR(18),  BNAME VARCHAR(18),  AMOUNT DECIMAL(8,2)  );  -- Insert data into DEPOSIT  INSERT INTO DEPOSIT (ACTNO, CNAME, BNAME, AMOUNT, ADATE) VALUES  ('A001', 'John Doe', 'Branch1', 5000.00, '1996-12-01'),  ('A002', 'Jane Smith', 'Branch2', 3500.00, '1996-12-15'),  ('A003', 'Alice Brown', 'Branch1', 6000.00, '1997-01-20');  -- Insert data into BRANCH  INSERT INTO BRANCH (BNAME, CITY) VALUES  ('Branch1', 'New York'),  ('Branch2', 'Los Angeles');  -- Insert data into CUSTOMERS  INSERT INTO CUSTOMERS (CNAME, CITY) VALUES  ('John Doe', 'New York'),  ('Jane Smith', 'Los Angeles'),  ('Alice Brown', 'San Francisco');  -- Insert data into BORROW  INSERT INTO BORROW (LOANNO, CNAME, BNAME, AMOUNT) VALUES  ('L001', 'John Doe', 'Branch1', 7000.00),  ('L002', 'Jane Smith', 'Branch2', 8000.00),  ('L003', 'Alice Brown', 'Branch1', 6500.00);  desc deposit;  desc branch;  Select \* from DEPOSIT;  Select \* from BORROW;  Select ACTNO,AMOUNT from DEPOSIT;  Select CNAME from DEPOSIT where AMOUNT>='4000';  Select CNAME from DEPOSIT where ADATE>='1996-12-01'; |
| 02 | Create the below given table and insert the data accordingly.  Job (job\_id, job\_title, min\_sal, max\_sal)  Employee (emp\_no, emp\_name, emp\_sal, emp\_comm, dept\_no)  deposit(a\_no,cname,bname,amount,a\_date).  borrow(loanno,cname,bname,amount).  Insert the data for all tables.  -> Perform following queries:  (1) Retrieve all data from employee, jobs and deposit.  (2) Give details of account no. and deposited rupees of customers having account opened between dates 01-01-06 and 25-07-06  (3) Display all jobs with minimum salary is greater than 4000  (4) Display name and salary of employee whose department no is 20. Give alias nameto name of employee.  (5) Display employee no, name and department details of those employee whose department lies in(10,20).  ->To study various options of LIKE predicate.  (1) Display all employee whose name start with ‘A’ and third character is ‘a’  (2) Display name, number and salary of those employees whose name is 5 characters long and first three characters are ‘Ani’  (3) Display the non-null values of employees and also employee name second character should be ‘n’ and string should be 5 characters long.  (4) Display the null values of employee and also employee name’s third character should be ‘a’.  (5) What will be output if you are giving LIKE predicate as ‘%\\_%’ ESCAPE ‘\’ |
|  | Create database Employee ;  use Employee;  create table Job (job\_id int , job\_title varchar(20), min\_sal int , max\_sal int);  create table Employee1 (emp\_no int , emp\_name varchar(30), emp\_sal float, emp\_comm varchar(10), dept\_no int);  create table deposit(a\_no int,cname varchar(50),bname varchar(50),amount float,a\_date date);  create table borrow(loanno varchar(50),cname varchar(50),bname varchar(50),amount int);  -- Insert data into Job  INSERT INTO Job (job\_id, job\_title, min\_sal, max\_sal) VALUES  (1, 'Manager', 5000, 12000),  (2, 'Developer', 3000, 8000),  (3, 'Analyst', 4000, 9000),  (4, 'Tester', 2500, 6000);  -- Insert data into Employee  INSERT INTO Employee1 (emp\_no, emp\_name, emp\_sal, emp\_comm, dept\_no) VALUES  (101, 'Alice', 6000, 500, 20),  (102, 'Bob', 7000, 700, 10),  (103, 'Charlie', 5000, 300, 20),  (104, 'David', 4000, NULL, 30),  (105, 'Eve', 4500, 400, 10);  -- Insert data into Deposit  INSERT INTO Deposit (a\_no, cname, bname, amount, a\_date) VALUES  (1, 'John Doe', 'Branch1', 1000, '2006-01-10'),  (2, 'Jane Smith', 'Branch2', 2000, '2006-06-15'),  (3, 'Alice Brown', 'Branch3', 1500, '2006-07-20'),  (4, 'Bob White', 'Branch4', 2500, '2005-12-25');  -- Insert data into Borrow  INSERT INTO Borrow (loanno, cname, bname, amount) VALUES  (1, 'John Doe', 'Branch1', 3000),  (2, 'Jane Smith', 'Branch2', 5000),  (3, 'Alice Brown', 'Branch3', 4500),  (4, 'Bob White', 'Branch4', 4000);  Select \* from employee1;  Select \* from jobs;  Select \* from deposit;  select a\_no,amount from deposit where a\_date between '01-01-06' and '25-07-06';  select \* from job where min\_sal>'4000';  SELECT emp\_name AS 'Employee Name', emp\_sal FROM Employee1 WHERE dept\_no = 20;  SELECT emp\_no, emp\_name, dept\_no FROM Employee1 WHERE dept\_no IN (10, 20);  SELECT \* FROM Employee1 WHERE emp\_name LIKE 'A\_a%';  SELECT emp\_name, emp\_no, emp\_sal FROM Employee1 WHERE emp\_name LIKE 'Ani\_\_';  SELECT emp\_name, emp\_no, emp\_sal FROM Employee1 WHERE emp\_name IS NOT NULL AND emp\_name LIKE '\_n\_\_\_';  SELECT emp\_no, emp\_name FROM Employee1 WHERE emp\_name IS NULL OR emp\_name LIKE '\_\_a%';  SELECT emp\_name FROM Employee1 WHERE emp\_name LIKE '%\\_%' ESCAPE '\'; |
| 03 | Create the below given table and insert the data accordingly.  Job (job\_id, job\_title, min\_sal, max\_sal)  Employee (emp\_no, emp\_name, emp\_sal, emp\_comm, dept\_no)  deposit(a\_no,cname,bname,amount,a\_date).  borrow(loanno,cname,bname,amount).  Insert the data for all tables.  To Perform various data manipulation commands, aggregate functions and sorting concept on all created tables.  (1) List total deposit from deposit. (2) List total loan from karolbagh branch (3) Give maximum loan from branch vrce. (4) Count total number of customers. (5) Count total number of customer’s cities (6) Create table supplier from employee with all the columns. (7) Create table sup1 from employee with first two columns. (8) Create table sup2 from employee with no data. (9) Insert the data into sup2 from employee whose second character should be ‘n’ and string should be 5 characters long in employee name field. (10) Delete all the rows from sup1**.** (11) Delete the detail of supplier whose sup\_no is 103  (12) Rename the table sup2  (13) Destroy table sup1 with all the data.  (14) Update the value dept\_no to 10 where second character of emp. name is ‘m’.  (15) Update the value of employee name whose employee number is 103. |
|  | CREATE TABLE Job ( job\_id INT,job\_title VARCHAR(50),min\_sal FLOAT,max\_sal FLOAT);  CREATE TABLE Employee (emp\_no INT, emp\_name VARCHAR(50), emp\_sal FLOAT, emp\_comm FLOAT,dept\_no INT);  CREATE TABLE Deposit (a\_no INT, cname VARCHAR(50),bname VARCHAR(50),amount float, a\_date DATE);  CREATE TABLE Borrow (loanno INT, cname VARCHAR(50),bname VARCHAR(50), amount DECIMAL(8,2));  INSERT INTO Job (job\_id, job\_title, min\_sal, max\_sal) VALUES  (1, 'Manager', 5000.00, 12000.00),  (2, 'Developer', 3000.00, 8000.00),  (3, 'Analyst', 4000.00, 9000.00),  (4, 'Tester', 2500.00, 6000.00);  INSERT INTO Employee (emp\_no, emp\_name, emp\_sal, emp\_comm, dept\_no) VALUES  (101, 'Alice', 6000.00, 500.00, 20),  (102, 'Bob', 7000.00, 700.00, 10),  (103, 'Charlie', 5000.00, 300.00, 20),  (104, 'David', 4000.00, NULL, 30),  (105, 'Eve', 4500.00, 400.00, 10);  INSERT INTO Deposit (a\_no, cname, bname, amount, a\_date) VALUES  (1, 'John Doe', 'KarolBagh', 1000.00, '2006-01-10'),  (2, 'Jane Smith', 'KarolBagh', 2000.00, '2006-06-15'),  (3, 'Alice Brown', 'VRCE', 1500.00, '2006-07-20'),  (4, 'Bob White', 'VRCE', 2500.00, '2005-12-25');  INSERT INTO Borrow (loanno, cname, bname, amount) VALUES  (1, 'John Doe', 'KarolBagh', 3000.00),  (2, 'Jane Smith', 'KarolBagh', 5000.00),  (3, 'Alice Brown', 'VRCE', 4500.00),  (4, 'Bob White', 'VRCE', 4000.00);  -- (1) List total deposit from Deposit  SELECT SUM(amount) AS Total\_Deposit FROM Deposit;  -- (2) List total loan from KarolBagh branch  SELECT SUM(amount) AS Total\_Loan\_KarolBagh FROM Borrow WHERE bname = 'KarolBagh';  -- (3) Give maximum loan from branch VRCE  SELECT MAX(amount) AS Max\_Loan\_VRCE FROM Borrow WHERE bname = 'VRCE';  -- (4) Count total number of customers  SELECT COUNT(DISTINCT cname) AS Total\_Customers FROM Deposit;  -- (5) Count total number of customers’ cities  SELECT COUNT(DISTINCT bname) AS Total\_Cities FROM Deposit;  -- (6) Create table supplier from employee with all the columns  CREATE TABLE Supplier AS SELECT \* FROM Employee;  -- (7) Create table sup1 from employee with first two columns  CREATE TABLE Sup1 AS SELECT emp\_no, emp\_name FROM Employee;  -- (8) Create table sup2 from employee with no data  CREATE TABLE Sup2 AS SELECT \* FROM Employee WHERE 1=0;  -- (9) Insert the data into Sup2 from employee whose second character is ‘n’ and name length is 5  INSERT INTO Sup2  SELECT \* FROM Employee  WHERE emp\_name LIKE '\_n\_\_\_';  -- (10) Delete all the rows from Sup1  DELETE FROM Sup1;  -- (11) Delete the detail of supplier whose emp\_no is 103  DELETE FROM Supplier WHERE emp\_no = 103;  -- (12) Rename the table Sup2  RENAME TABLE Sup2 TO RenamedSup2;  -- (13) Destroy table Sup1 with all the data  DROP TABLE Sup1;  -- (14) Update dept\_no to 10 where second character of emp\_name is ‘m’  UPDATE Employee  SET dept\_no = 10  WHERE emp\_name LIKE '\_m%';  -- (15) Update the value of emp\_name where emp\_no is 103  UPDATE Employee  SET emp\_name = 'UpdatedName'  WHERE emp\_no = 103; |
| 04 | To study Single-row functions.  (i)Create tables according to the need.  (ii) Insert the data for all tables.  (1) Write a query to display the current date. Label the column Date.  (2) For each employee, display the employee number, job, salary, and salary increased by 15% and expressed as a whole number. Label the column New Salary  (3) Modify your query to add a column that subtracts the old salary from the new salary. Label the column Increase  (4) Write a query that displays the employee’s names with the first letter capitalized and all other letters lowercase, and the length of the names, for all employees whose name starts with J, A, or M. Give each column an appropriate label. Sort the results by the employees’ last names  (5) Write a query that produces the following for each employee: earns monthly  (6) Display the name, hire date, number of months employed and day of the week on which the employee has started. Order the results by the day of the week starting with Monday  (7) Display the hiredate of emp in a format that appears as Seventh of June 1994 12:00:00 AM  (8) Write a query to calculate the annual compensation of all employees (sal+comm.). |
|  | -- Step 1: Create Employee table  CREATE TABLE Employee (  emp\_no INT,  emp\_name VARCHAR(50),  job VARCHAR(50),  salary DECIMAL(8,2),  comm DECIMAL(8,2),  hire\_date DATE  );  -- Step 2: Insert data into Employee table  INSERT INTO Employee (emp\_no, emp\_name, job, salary, comm, hire\_date) VALUES  (101, 'John', 'Manager', 6000.00, 500.00, '2010-06-07'),  (102, 'Alice', 'Developer', 4500.00, NULL, '2015-08-15'),  (103, 'Michael', 'Analyst', 5200.00, 300.00, '2012-09-10'),  (104, 'James', 'Tester', 3500.00, 200.00, '2018-01-25'),  (105, 'Mary', 'HR', 4800.00, 400.00, '2016-12-03');  -- Step 3: Queries  -- (1) Display the current date  SELECT CURDATE() AS 'Date';  -- (2) Display employee number, job, salary, and salary increased by 15% as a whole number  SELECT  emp\_no,  job,  salary,  ROUND(salary \* 1.15) AS 'New Salary'  FROM Employee;  -- (3) Add a column showing the increase in salary  SELECT  emp\_no,  job,  salary,  ROUND(salary \* 1.15) AS 'New Salary',  ROUND(salary \* 1.15) - salary AS 'Increase'  FROM Employee;  -- (4) Display employee names with the first letter capitalized, name length, and names starting with J, A, or M  SELECT  CONCAT(UCASE(LEFT(emp\_name, 1)), LCASE(SUBSTRING(emp\_name, 2))) AS 'Name',  LENGTH(emp\_name) AS 'Name Length'  FROM Employee  WHERE emp\_name LIKE 'J%' OR emp\_name LIKE 'A%' OR emp\_name LIKE 'M%'  ORDER BY emp\_name;  -- (5) Display monthly earnings for each employee  SELECT  emp\_name AS 'Employee Name',  ROUND(salary / 12, 2) AS 'Monthly Earnings'  FROM Employee;  -- (6) Display name, hire date, number of months employed, and the day of the week of hire  SELECT  emp\_name AS 'Employee Name',  hire\_date AS 'Hire Date',  TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) AS 'Months Employed',  DAYNAME(hire\_date) AS 'Day of Week'  FROM Employee  ORDER BY FIELD(DAYNAME(hire\_date), 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday');  -- (7) Display hire date in the format: "Seventh of June 1994 12:00:00 AM"  SELECT  emp\_name AS 'Employee Name',  DATE\_FORMAT(hire\_date, '%D of %M %Y %r') AS 'Formatted Hire Date'  FROM Employee;  -- (8) Calculate the annual compensation of all employees (salary + commission)  SELECT  emp\_name AS 'Employee Name',  (salary + IFNULL(comm, 0)) AS 'Annual Compensation'  FROM Employee; |
| 05 | Displaying data from Multiple Tables (join)  (1) Give details of customers ANIL.  (2) Give name of customer who are borrowers and depositors and having living city Nagpur  (3) Give city as their city name of customers having same living branch.  (4) Write a query to display the last name, department number, and department name for all employees.  (5) Create a unique listing of all jobs that are in department 30. Include the location of the department in the output  (6) Write a query to display the employee name, department number, and department name for all employees who work in NEW YORK.  (7) Display the employee last name and employee number along with their manager’s last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively  (8) Create a query to display the name and hire date of any employee hired after employee SCOTT |
|  | -- Step 1: Create Employee table  CREATE TABLE Employee (  emp\_no INT,  emp\_name VARCHAR(50),  job VARCHAR(50),  salary DECIMAL(8,2),  comm DECIMAL(8,2),  hire\_date DATE  );  -- Step 2: Insert data into Employee table  INSERT INTO Employee (emp\_no, emp\_name, job, salary, comm, hire\_date) VALUES  (101, 'John', 'Manager', 6000.00, 500.00, '2010-06-07'),  (102, 'Alice', 'Developer', 4500.00, NULL, '2015-08-15'),  (103, 'Michael', 'Analyst', 5200.00, 300.00, '2012-09-10'),  (104, 'James', 'Tester', 3500.00, 200.00, '2018-01-25'),  (105, 'Mary', 'HR', 4800.00, 400.00, '2016-12-03');  -- Step 3: Queries  -- (1) Display the current date  SELECT CURDATE() AS 'Date';  -- (2) Display employee number, job, salary, and salary increased by 15% as a whole number  SELECT  emp\_no,  job,  salary,  ROUND(salary \* 1.15) AS 'New Salary'  FROM Employee;  -- (3) Add a column showing the increase in salary  SELECT  emp\_no,  job,  salary,  ROUND(salary \* 1.15) AS 'New Salary',  ROUND(salary \* 1.15) - salary AS 'Increase'  FROM Employee;  -- (4) Display employee names with the first letter capitalized, name length, and names starting with J, A, or M  SELECT  CONCAT(UCASE(LEFT(emp\_name, 1)), LCASE(SUBSTRING(emp\_name, 2))) AS 'Name',  LENGTH(emp\_name) AS 'Name Length'  FROM Employee  WHERE emp\_name LIKE 'J%' OR emp\_name LIKE 'A%' OR emp\_name LIKE 'M%'  ORDER BY emp\_name;  -- (5) Display monthly earnings for each employee  SELECT  emp\_name AS 'Employee Name',  ROUND(salary / 12, 2) AS 'Monthly Earnings'  FROM Employee;  -- (6) Display name, hire date, number of months employed, and the day of the week of hire  SELECT  emp\_name AS 'Employee Name',  hire\_date AS 'Hire Date',  TIMESTAMPDIFF(MONTH, hire\_date, CURDATE()) AS 'Months Employed',  DAYNAME(hire\_date) AS 'Day of Week'  FROM Employee  ORDER BY FIELD(DAYNAME(hire\_date), 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday');  -- (7) Display hire date in the format: "Seventh of June 1994 12:00:00 AM"  SELECT  emp\_name AS 'Employee Name',  DATE\_FORMAT(hire\_date, '%D of %M %Y %r') AS 'Formatted Hire Date'  FROM Employee;  -- (8) Calculate the annual compensation of all employees (salary + commission)  SELECT  emp\_name AS 'Employee Name',  (salary + IFNULL(comm, 0)) AS 'Annual Compensation'  FROM Employee; |
| 06 | To apply the concept of Aggregating Data using Group functions  (1) List total deposit of customer having account date after 1-jan-96  (2) List total deposit of customers living in city Nagpur  (3) List maximum deposit of customers living in bombay.  (4) Display the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number.  (5) Write a query that displays the difference between the highest and lowest salaries. Label the column DIFFERENCE.  (6) Create a query that will display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998.  (7) Find the average salaries for each department without displaying the respective department numbers.  (8) Write a query to display the total salary being paid to each job title, within each department  (9) Find the average salaries > 2000 for each department without displaying the respective department numbers.  (10) Display the job and total salary for each job with a total salary amount exceeding 3000, in which excludes president and sorts the list by the total salary  (11) List the branches having sum of deposit more than 5000 and located in city bombay. |
|  | -- Step 1: Create necessary tables  -- Employee Table  CREATE TABLE Employee (  emp\_no INT,  emp\_name VARCHAR(50),  job VARCHAR(50),  dept\_no INT,  salary DECIMAL(8, 2),  hire\_date DATE  );  -- Customer Table  CREATE TABLE Customer (  cname VARCHAR(50),  city VARCHAR(50),  branch\_name VARCHAR(50)  );  -- Deposit Table  CREATE TABLE Deposit (  a\_no INT,  cname VARCHAR(50),  bname VARCHAR(50),  amount DECIMAL(8, 2),  a\_date DATE  );  -- Insert sample data into tables  -- Insert data into Employee table  INSERT INTO Employee (emp\_no, emp\_name, job, dept\_no, salary, hire\_date) VALUES  (101, 'John', 'Manager', 10, 5000.00, '1995-01-15'),  (102, 'Alice', 'Analyst', 20, 4000.00, '1996-07-23'),  (103, 'Bob', 'Clerk', 10, 3000.00, '1997-03-19'),  (104, 'James', 'President', 30, 7000.00, '1995-05-11'),  (105, 'Mary', 'Developer', 20, 4500.00, '1998-10-10');  -- Insert data into Customer table  INSERT INTO Customer (cname, city, branch\_name) VALUES  ('Anil', 'Nagpur', 'Branch1'),  ('Ravi', 'Bombay', 'Branch2'),  ('Priya', 'Delhi', 'Branch1');  -- Insert data into Deposit table  INSERT INTO Deposit (a\_no, cname, bname, amount, a\_date) VALUES  (1, 'Anil', 'Branch1', 2000.00, '1997-01-10'),  (2, 'Ravi', 'Branch2', 6000.00, '1999-06-15'),  (3, 'Priya', 'Branch1', 1500.00, '1995-12-20');  -- Step 2: Queries  -- (1) List total deposit of customers having account date after 1-Jan-96  SELECT  SUM(amount) AS 'Total Deposit'  FROM Deposit  WHERE a\_date > '1996-01-01';  -- (2) List total deposit of customers living in city Nagpur  SELECT  SUM(d.amount) AS 'Total Deposit'  FROM Deposit d  JOIN Customer c ON d.cname = c.cname  WHERE c.city = 'Nagpur';  -- (3) List maximum deposit of customers living in Bombay  SELECT  MAX(d.amount) AS 'Maximum Deposit'  FROM Deposit d  JOIN Customer c ON d.cname = c.cname  WHERE c.city = 'Bombay';  -- (4) Display highest, lowest, sum, and average salary of all employees  SELECT  ROUND(MAX(salary)) AS 'Maximum',  ROUND(MIN(salary)) AS 'Minimum',  ROUND(SUM(salary)) AS 'Sum',  ROUND(AVG(salary)) AS 'Average'  FROM Employee;  -- (5) Display the difference between the highest and lowest salaries  SELECT  MAX(salary) - MIN(salary) AS 'DIFFERENCE'  FROM Employee;  -- (6) Display total employees and employees hired in 1995, 1996, 1997, and 1998  SELECT  COUNT(\*) AS 'Total Employees',  SUM(CASE WHEN YEAR(hire\_date) = 1995 THEN 1 ELSE 0 END) AS 'Hired in 1995',  SUM(CASE WHEN YEAR(hire\_date) = 1996 THEN 1 ELSE 0 END) AS 'Hired in 1996',  SUM(CASE WHEN YEAR(hire\_date) = 1997 THEN 1 ELSE 0 END) AS 'Hired in 1997',  SUM(CASE WHEN YEAR(hire\_date) = 1998 THEN 1 ELSE 0 END) AS 'Hired in 1998'  FROM Employee;  -- (7) Find average salaries for each department without displaying department numbers  SELECT  AVG(salary) AS 'Average Salary'  FROM Employee  GROUP BY dept\_no;  -- (8) Total salary being paid to each job title within each department  SELECT  job,  dept\_no,  SUM(salary) AS 'Total Salary'  FROM Employee  GROUP BY job, dept\_no;  -- (9) Find average salaries > 2000 for each department without displaying department numbers  SELECT  AVG(salary) AS 'Average Salary'  FROM Employee  WHERE salary > 2000  GROUP BY dept\_no;  -- (10) Display job and total salary for each job where total salary > 3000, excluding 'President'  SELECT  job,  SUM(salary) AS 'Total Salary'  FROM Employee  WHERE job != 'President'  GROUP BY job  HAVING SUM(salary) > 3000  ORDER BY SUM(salary) DESC;  -- (11) List branches having sum of deposits > 5000 and located in Bombay  SELECT  d.bname AS 'Branch',  SUM(d.amount) AS 'Total Deposit'  FROM Deposit d  JOIN Customer c ON d.bname = c.branch\_name  WHERE c.city = 'Bombay'  GROUP BY d.bname  HAVING SUM(d.amount) > 5000; |
| 07 | To solve queries using the concept of sub query.  (1) Write a query to display the last name and hire date of any employee in the same department as SCOTT. Exclude SCOTT  (2) Give name of customers who are depositors having same branch city of mr. sunil  (3) Give deposit details and loan details of customer in same city where pramod is living.  (4) Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in ascending order of salary  (5) Give names of depositors having same living city as mr. anil and having deposit amount greater than 2000  (6) Display the last name and salary of every employee who reports to ford.  (7) Display the department number, name, and job for every employee in the Accounting department.  (8) List the name of branch having highest number of depositors  (9) Give the name of cities where in which the maximum numbers of branches are located.  (10) Give the name of customers living in same city where maximum depositors are located. |
|  | -- Step 1: Create necessary tables  -- Employee Table  CREATE TABLE Employee (  emp\_no INT,  emp\_name VARCHAR(50),  job VARCHAR(50),  dept\_no INT,  salary DECIMAL(8, 2),  hire\_date DATE,  mgr\_id INT  );  -- Customer Table  CREATE TABLE Customer (  cname VARCHAR(50),  city VARCHAR(50),  branch\_name VARCHAR(50)  );  -- Deposit Table  CREATE TABLE Deposit (  a\_no INT,  cname VARCHAR(50),  bname VARCHAR(50),  amount DECIMAL(8, 2),  a\_date DATE  );  -- Borrow Table  CREATE TABLE Borrow (  loanno INT,  cname VARCHAR(50),  bname VARCHAR(50),  amount DECIMAL(8, 2)  );  -- Insert sample data into tables  -- Insert data into Employee table  INSERT INTO Employee (emp\_no, emp\_name, job, dept\_no, salary, hire\_date, mgr\_id) VALUES  (101, 'John', 'Manager', 10, 5000.00, '1995-01-15', NULL),  (102, 'Alice', 'Analyst', 20, 4000.00, '1996-07-23', 101),  (103, 'Bob', 'Clerk', 10, 3000.00, '1997-03-19', 101),  (104, 'James', 'President', 30, 7000.00, '1995-05-11', NULL),  (105, 'Mary', 'Developer', 20, 4500.00, '1998-10-10', 102);  -- Insert data into Customer table  INSERT INTO Customer (cname, city, branch\_name) VALUES  ('Anil', 'Nagpur', 'Branch1'),  ('Ravi', 'Bombay', 'Branch2'),  ('Priya', 'Delhi', 'Branch1'),  ('Sunil', 'Nagpur', 'Branch3'),  ('Pramod', 'Delhi', 'Branch1');  -- Insert data into Deposit table  INSERT INTO Deposit (a\_no, cname, bname, amount, a\_date) VALUES  (1, 'Anil', 'Branch1', 2000.00, '1997-01-10'),  (2, 'Ravi', 'Branch2', 6000.00, '1999-06-15'),  (3, 'Priya', 'Branch1', 1500.00, '1995-12-20'),  (4, 'Sunil', 'Branch3', 2500.00, '1997-07-15'),  (5, 'Pramod', 'Branch1', 3500.00, '1998-08-20');  -- Insert data into Borrow table  INSERT INTO Borrow (loanno, cname, bname, amount) VALUES  (1, 'Anil', 'Branch1', 5000.00),  (2, 'Ravi', 'Branch2', 7000.00),  (3, 'Pramod', 'Branch1', 4000.00);  -- Step 2: Queries using subqueries  -- (1) Display the last name and hire date of any employee in the same department as SCOTT. Exclude SCOTT  SELECT emp\_name, hire\_date  FROM Employee  WHERE dept\_no = (SELECT dept\_no FROM Employee WHERE emp\_name = 'SCOTT')  AND emp\_name != 'SCOTT';  -- (2) Give the name of customers who are depositors having the same branch city as Mr. Sunil  SELECT cname  FROM Customer  WHERE branch\_name IN (  SELECT branch\_name FROM Customer WHERE cname = 'Sunil'  )  AND cname IN (  SELECT cname FROM Deposit WHERE amount > 0  );  -- (3) Give deposit details and loan details of customers in the same city where Pramod is living  SELECT d.a\_no, d.cname, d.bname, d.amount AS 'Deposit Amount', b.loanno, b.amount AS 'Loan Amount'  FROM Deposit d  LEFT JOIN Borrow b ON d.cname = b.cname  WHERE d.cname IN (  SELECT cname FROM Customer WHERE city = (  SELECT city FROM Customer WHERE cname = 'Pramod'  )  );  -- (4) Display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in ascending order of salary  SELECT emp\_no, emp\_name  FROM Employee  WHERE salary > (SELECT AVG(salary) FROM Employee)  ORDER BY salary ASC;  -- (5) Give names of depositors having the same living city as Mr. Anil and having deposit amount greater than 2000  SELECT cname  FROM Customer  WHERE city = (  SELECT city FROM Customer WHERE cname = 'Anil'  )  AND cname IN (  SELECT cname FROM Deposit WHERE amount > 2000  );  -- (6) Display the last name and salary of every employee who reports to Ford  SELECT emp\_name, salary  FROM Employee  WHERE mgr\_id = (SELECT emp\_no FROM Employee WHERE emp\_name = 'Ford');  -- (7) Display the department number, name, and job for every employee in the Accounting department  SELECT dept\_no, job  FROM Employee  WHERE dept\_no = (SELECT dept\_no FROM Employee WHERE job = 'Accounting');  -- (8) List the name of the branch having the highest number of depositors  SELECT branch\_name  FROM Customer  GROUP BY branch\_name  HAVING COUNT(cname) = (  SELECT MAX(depositor\_count)  FROM (SELECT COUNT(cname) AS depositor\_count FROM Customer GROUP BY branch\_name) AS branch\_counts  );  -- (9) Give the name of cities in which the maximum number of branches are located  SELECT city  FROM Customer  GROUP BY city  HAVING COUNT(DISTINCT branch\_name) = (  SELECT MAX(branch\_count)  FROM (SELECT city, COUNT(DISTINCT branch\_name) AS branch\_count FROM Customer GROUP BY city) AS city\_counts  );  -- (10) Give the name of customers living in the same city where maximum depositors are located  SELECT cname  FROM Customer  WHERE city = (  SELECT city  FROM Customer  GROUP BY city  HAVING COUNT(cname) = (  SELECT MAX(city\_count)  FROM (SELECT city, COUNT(cname) AS city\_count FROM Customer GROUP BY city) AS city\_counts  )  ); |
| 08 | Manipulating Data  (1) Give 10% interest to all depositors.  (2) Give 10% interest to all depositors having branch vrce.  (3) Give 10% interest to all depositors living in nagpur and having branch city bombay.  (4) Write a query which changes the department number of all employees with empno7788’s job to employee 7844’current department number  (5) Write a query which changes the department number of all employees with empno.  (6) Transfer 10 Rs from account of anil to sunil if both are having same branch  (7) Give 100 Rs more to all depositors if they are maximum depositors in their respective branch.  (8) Delete deposit of vijay  (9) Delete borrower of branches having average loan less than 1000 |
|  | -- Step 1: Create the necessary tables if not already created  -- Employee Table  CREATE TABLE Employee (  emp\_no INT,  emp\_name VARCHAR(50),  job VARCHAR(50),  dept\_no INT,  salary DECIMAL(8, 2),  hire\_date DATE,  mgr\_id INT  );  -- Customer Table  CREATE TABLE Customer (  cname VARCHAR(50),  city VARCHAR(50),  branch\_name VARCHAR(50)  );  -- Deposit Table  CREATE TABLE Deposit (  a\_no INT,  cname VARCHAR(50),  bname VARCHAR(50),  amount DECIMAL(8, 2),  a\_date DATE  );  -- Borrow Table  CREATE TABLE Borrow (  loanno INT,  cname VARCHAR(50),  bname VARCHAR(50),  amount DECIMAL(8, 2)  );  -- Insert sample data into tables (if not inserted already)  -- Insert data into Employee table  INSERT INTO Employee (emp\_no, emp\_name, job, dept\_no, salary, hire\_date, mgr\_id) VALUES  (101, 'John', 'Manager', 10, 5000.00, '1995-01-15', NULL),  (102, 'Alice', 'Analyst', 20, 4000.00, '1996-07-23', 101),  (103, 'Bob', 'Clerk', 10, 3000.00, '1997-03-19', 101),  (104, 'James', 'President', 30, 7000.00, '1995-05-11', NULL),  (105, 'Mary', 'Developer', 20, 4500.00, '1998-10-10', 102);  -- Insert data into Customer table  INSERT INTO Customer (cname, city, branch\_name) VALUES  ('Anil', 'Nagpur', 'Branch1'),  ('Ravi', 'Bombay', 'Branch2'),  ('Priya', 'Delhi', 'Branch1'),  ('Sunil', 'Nagpur', 'Branch3'),  ('Pramod', 'Delhi', 'Branch1');  -- Insert data into Deposit table  INSERT INTO Deposit (a\_no, cname, bname, amount, a\_date) VALUES  (1, 'Anil', 'Branch1', 2000.00, '1997-01-10'),  (2, 'Ravi', 'Branch2', 6000.00, '1999-06-15'),  (3, 'Priya', 'Branch1', 1500.00, '1995-12-20'),  (4, 'Sunil', 'Branch3', 2500.00, '1997-07-15'),  (5, 'Pramod', 'Branch1', 3500.00, '1998-08-20');  -- Insert data into Borrow table  INSERT INTO Borrow (loanno, cname, bname, amount) VALUES  (1, 'Anil', 'Branch1', 5000.00),  (2, 'Ravi', 'Branch2', 7000.00),  (3, 'Pramod', 'Branch1', 4000.00);  -- Step 2: Queries for data manipulation  -- (1) Give 10% interest to all depositors  UPDATE Deposit  SET amount = amount \* 1.10;  -- (2) Give 10% interest to all depositors having branch 'vrce'  UPDATE Deposit  SET amount = amount \* 1.10  WHERE bname = 'vrce';  -- (3) Give 10% interest to all depositors living in Nagpur and having branch city Bombay  UPDATE Deposit  SET amount = amount \* 1.10  WHERE cname IN (SELECT cname FROM Customer WHERE city = 'Nagpur')  AND bname = 'Bombay';  -- (4) Change the department number of all employees with emp\_no 7788’s job to employee 7844’s current department number  UPDATE Employee  SET dept\_no = (SELECT dept\_no FROM Employee WHERE emp\_no = 7844)  WHERE job = (SELECT job FROM Employee WHERE emp\_no = 7788);  -- (5) Change the department number of all employees with emp\_no  UPDATE Employee  SET dept\_no = 10 -- Assign the desired department number here  WHERE emp\_no = 7788;  -- (6) Transfer 10 Rs from Anil’s account to Sunil’s account if both are having the same branch  UPDATE Deposit  SET amount = amount - 10  WHERE cname = 'Anil' AND bname = 'Branch3';  UPDATE Deposit  SET amount = amount + 10  WHERE cname = 'Sunil' AND bname = 'Branch3';  -- (7) Give 100 Rs more to all depositors if they are maximum depositors in their respective branch  UPDATE Deposit  SET amount = amount + 100  WHERE cname IN (  SELECT cname  FROM Deposit  WHERE bname IN (  SELECT bname  FROM Deposit  GROUP BY bname  HAVING COUNT(cname) = (  SELECT MAX(count\_depositors)  FROM (  SELECT bname, COUNT(cname) AS count\_depositors  FROM Deposit  GROUP BY bname  ) AS branch\_deposits  )  )  );  -- (8) Delete deposit of Vijay  DELETE FROM Deposit  WHERE cname = 'Vijay';  -- (9) Delete borrower of branches having average loan less than 1000  DELETE FROM Borrow  WHERE bname IN (  SELECT bname  FROM Borrow  GROUP BY bname  HAVING AVG(amount) < 1000  ); |
| 09 | To Perform Operations Using PL/SQL.  1. Write a PL/SQL Block that will get the salary of employee with employee number ‘105’ and display it on the Screen  2. Write a PL/SQL block that demonstrates use of CONSTANT.  3. Write a PL/SQL Block that demonstrates Decision making Statements.  4. Write a PL/SQL block that prints 1 to 5 numbers Using LOOP…EXIT WHEN Statement.  5. Write a PL/SQL block that prints 1 to 5 numbers Using WHILE Loop Statement |
|  | -- Step 1: Create Database and Tables  CREATE DATABASE EmployeeDB;  USE EmployeeDB;  -- Create Employee Table  CREATE TABLE Employees (  EmployeeID INT PRIMARY KEY,  EmployeeName VARCHAR(100),  Salary DECIMAL(10, 2)  );  -- Insert sample data  INSERT INTO Employees (EmployeeID, EmployeeName, Salary) VALUES  (101, 'John Doe', 50000),  (102, 'Jane Smith', 60000),  (103, 'Alice Johnson', 55000),  (104, 'Robert Brown', 45000),  (105, 'Emily Davis', 70000);  -- Step 2: Write PL/SQL Blocks  DELIMITER $$  -- 1. Get the salary of employee with employee number '105' and display it on the screen  CREATE PROCEDURE GetEmployeeSalary()  BEGIN  DECLARE empSalary DECIMAL(10, 2);  SELECT Salary INTO empSalary FROM Employees WHERE EmployeeID = 105;  SELECT CONCAT('Salary of Employee 105: ', empSalary) AS Result;  END$$  -- 2. Demonstrate the use of CONSTANT  CREATE PROCEDURE UseConstant()  BEGIN  DECLARE CONSTANT\_VAR VARCHAR(50) DEFAULT 'This is a constant value.';  SELECT CONSTANT\_VAR AS Result;  END$$  -- 3. Demonstrate Decision Making Statements  CREATE PROCEDURE DecisionMakingDemo()  BEGIN  DECLARE salary DECIMAL(10, 2);  SELECT Salary INTO salary FROM Employees WHERE EmployeeID = 105;  IF salary > 60000 THEN  SELECT 'Employee 105 earns more than 60000' AS Result;  ELSE  SELECT 'Employee 105 earns less than or equal to 60000' AS Result;  END IF;  END$$  -- 4. Print 1 to 5 numbers using LOOP…EXIT WHEN Statement  CREATE PROCEDURE LoopExitWhen()  BEGIN  DECLARE counter INT DEFAULT 1;  my\_loop: LOOP  SELECT counter AS Number;  SET counter = counter + 1;  IF counter > 5 THEN  LEAVE my\_loop;  END IF;  END LOOP my\_loop;  END$$  -- 5. Print 1 to 5 numbers using WHILE Loop Statement  CREATE PROCEDURE WhileLoopDemo()  BEGIN  DECLARE counter INT DEFAULT 1;  WHILE counter <= 5 DO  SELECT counter AS Number;  SET counter = counter + 1;  END WHILE;  END$$  DELIMITER ;  -- Step 3: Call Procedures to Test  CALL GetEmployeeSalary();  CALL UseConstant();  CALL DecisionMakingDemo();  CALL LoopExitWhen();  CALL WhileLoopDemo(); |
| 10 | 6. Write a PL/SQL block that prints 1 to 5 numbers Using FOR Loop Statement.  7. Write a PL/SQL block that demonstrates Use of SQL Statements inside PL/SQL BLOCK  8. Write a PL/SQL block that implements Implicit Cursor.  9. Write a PL/SQL block that implements Explicit Cursor.  10. Write a PL/SQL block that implements Stored Procedure with IN, OUT, INOUT Parameters with EXPEPTION HANDLING mechanism |
|  | -- Step 1: Create Database and Tables  CREATE DATABASE EmployeeDB;  USE EmployeeDB;  -- Create Employee Table  CREATE TABLE Employees (  EmployeeID INT PRIMARY KEY,  EmployeeName VARCHAR(100),  Salary DECIMAL(10, 2)  );  -- Insert sample data  INSERT INTO Employees (EmployeeID, EmployeeName, Salary) VALUES  (101, 'John Doe', 50000),  (102, 'Jane Smith', 60000),  (103, 'Alice Johnson', 55000),  (104, 'Robert Brown', 45000),  (105, 'Emily Davis', 70000);  -- Step 2: Write PL/SQL Blocks  DELIMITER $$  -- 6. Print 1 to 5 numbers using FOR Loop Statement  CREATE PROCEDURE ForLoopDemo()  BEGIN  DECLARE counter INT;  FOR counter IN 1..5 DO  SELECT counter AS Number;  END FOR;  END$$  -- 7. Demonstrate Use of SQL Statements inside PL/SQL BLOCK  CREATE PROCEDURE SqlInsidePlsqlDemo()  BEGIN  DECLARE totalSalary DECIMAL(10, 2);  SELECT SUM(Salary) INTO totalSalary FROM Employees;  SELECT CONCAT('Total Salary of all employees: ', totalSalary) AS Result;  END$$  -- 8. Implement Implicit Cursor  CREATE PROCEDURE ImplicitCursorDemo()  BEGIN  DECLARE avgSalary DECIMAL(10, 2);  SELECT AVG(Salary) INTO avgSalary FROM Employees;  SELECT CONCAT('Average Salary of Employees: ', avgSalary) AS Result;  END$$  -- 9. Implement Explicit Cursor  CREATE PROCEDURE ExplicitCursorDemo()  BEGIN  DECLARE empName VARCHAR(100);  DECLARE empSalary DECIMAL(10, 2);  DECLARE done INT DEFAULT FALSE;  DECLARE empCursor CURSOR FOR SELECT EmployeeName, Salary FROM Employees;  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  OPEN empCursor;  read\_loop: LOOP  FETCH empCursor INTO empName, empSalary;  IF done THEN  LEAVE read\_loop;  END IF;  SELECT CONCAT('Employee: ', empName, ', Salary: ', empSalary) AS Result;  END LOOP;  CLOSE empCursor;  END$$  -- 10. Implement Stored Procedure with IN, OUT, INOUT Parameters and Exception Handling  CREATE PROCEDURE EmployeeDetails(IN empID INT, OUT empName VARCHAR(100), INOUT empSalary DECIMAL(10, 2))  BEGIN  DECLARE empFound INT DEFAULT 0;  -- Try to fetch details  BEGIN  SELECT COUNT(\*) INTO empFound FROM Employees WHERE EmployeeID = empID;  IF empFound = 0 THEN  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT = 'Employee not found!';  ELSE  SELECT EmployeeName, Salary INTO empName, empSalary FROM Employees WHERE EmployeeID = empID;  SET empSalary = empSalary + 1000; -- Example modification for INOUT  END IF;  END;  EXCEPTION  WHEN SQLSTATE '45000' THEN  SELECT 'Error: Employee not found!' AS Error;  END$$  DELIMITER ;  -- Step 3: Call Procedures to Test  CALL ForLoopDemo();  CALL SqlInsidePlsqlDemo();  CALL ImplicitCursorDemo();  CALL ExplicitCursorDemo();  -- Test Procedure with IN, OUT, INOUT Parameters  SET @empName = NULL;  SET @empSalary = 0;  CALL EmployeeDetails(105, @empName, @empSalary);  SELECT @empName AS EmployeeName, @empSalary AS UpdatedSalary; |
| 11 | 11. Write a PL/SQL block that implements Function.  12. Write a PL/SQL block that implements AFTER UPDATE TRIGGER.  13. Write a PL/SQL block that implements BEFORE UPDATE TRIGGER  14. Write a trigger to check the mark is not Zero or Negative.  15. Write a Trigger that check the employee name must starts with ‘M’ |
|  | -- Step 1: Create Database and Tables  CREATE DATABASE EmployeeDB;  USE EmployeeDB;  -- Create Employee Table  CREATE TABLE Employees (  EmployeeID INT PRIMARY KEY,  EmployeeName VARCHAR(100),  Salary DECIMAL(10, 2)  );  -- Insert sample data  INSERT INTO Employees (EmployeeID, EmployeeName, Salary) VALUES  (101, 'John Doe', 50000),  (102, 'Mary Smith', 60000),  (103, 'Michael Johnson', 55000),  (104, 'Robert Brown', 45000),  (105, 'Emily Davis', 70000);  -- Create Marks Table  CREATE TABLE Marks (  StudentID INT PRIMARY KEY,  StudentName VARCHAR(100),  Marks INT  );  -- Insert sample data for Marks  INSERT INTO Marks (StudentID, StudentName, Marks) VALUES  (1, 'Alice', 85),  (2, 'Bob', 90),  (3, 'Charlie', -5); -- Example invalid value  -- Step 2: Write PL/SQL Blocks  DELIMITER $$  -- 11. Function Implementation  CREATE FUNCTION CalculateBonus(salary DECIMAL(10, 2)) RETURNS DECIMAL(10, 2)  BEGIN  RETURN salary \* 0.10; -- Calculate 10% bonus  END$$  -- 12. AFTER UPDATE Trigger  CREATE TRIGGER AfterSalaryUpdate  AFTER UPDATE ON Employees  FOR EACH ROW  BEGIN  INSERT INTO Marks (StudentID, StudentName, Marks)  VALUES (NEW.EmployeeID, CONCAT('Updated: ', NEW.EmployeeName), NEW.Salary \* 0.01);  END$$  -- 13. BEFORE UPDATE Trigger  CREATE TRIGGER BeforeSalaryUpdate  BEFORE UPDATE ON Employees  FOR EACH ROW  BEGIN  IF NEW.Salary < 0 THEN  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT = 'Salary cannot be negative!';  END IF;  END$$  -- 14. Trigger to check mark is not zero or negative  CREATE TRIGGER CheckValidMarks  BEFORE INSERT ON Marks  FOR EACH ROW  BEGIN  IF NEW.Marks <= 0 THEN  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT = 'Marks cannot be zero or negative!';  END IF;  END$$  -- 15. Trigger to ensure employee name starts with ‘M’  CREATE TRIGGER CheckEmployeeName  BEFORE INSERT ON Employees  FOR EACH ROW  BEGIN  IF LEFT(NEW.EmployeeName, 1) <> 'M' THEN  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT = 'Employee name must start with "M"!';  END IF;  END$$  DELIMITER ;  -- Step 3: Test Implementations  -- Test Function  SELECT CalculateBonus(60000) AS Bonus;  -- Test Triggers  -- Test AFTER UPDATE Trigger  UPDATE Employees SET Salary = 75000 WHERE EmployeeID = 105;  -- Test BEFORE UPDATE Trigger  UPDATE Employees SET Salary = -1000 WHERE EmployeeID = 105; -- Should throw an error  -- Test Marks Trigger  INSERT INTO Marks (StudentID, StudentName, Marks) VALUES (4, 'Diana', -10); -- Should throw an error  -- Test Employee Name Trigger  INSERT INTO Employees (EmployeeID, EmployeeName, Salary) VALUES (106, 'Alice', 50000); -- Should throw an error |
| 12 | Consider the following relations:  Student(snum: integer, sname: string, major: string, level: string, age: integer)  Class(name: string, meets at: string, room: string, fid: integer)  Enrolled(snum: integer, cname: string)  Faculty(fid: integer, fname: string, deptid: integer)  The meaning of these relations is straightforward; for example, Enrolled has one record  per student-class pair such that the student is enrolled in the class.  Write the following queries in SQL. No duplicates should be printed in any of the  answers.  1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by  I. Teach.  2. Find the age of the oldest student who is either a History major or enrolled in a  course taught by I. Teach.  3. Find the names of all classes that either meet in room R128 or have five or more  students enrolled.  4. Find the names of all students who are enrolled in two classes that meet at the  same time  5. Find the names of faculty members who teach in every room in which some class  is taught.  6. Find the names of faculty members for whom the combined enrollment of the  courses that they teach is less than five.  7. For each level, print the level and the average age of students for that level.  8. For all levels except JR, print the level and the average age of students for that  level.  9. For each faculty member that has taught classes only in room R128, print the  faculty member’s name and the total number of classes she or he has taught.  10. Find the names of students enrolled in the maximum number of classes.  11. Find the names of students not enrolled in any class.  12. For each age value that appears in Students, find the level value that appears most  often. For example, if there are more FR level students aged 18 than SR, JR, or  SO students aged 18, you should print the pair (18, FR). |
|  | -- Step 1: Create Database and Tables  CREATE DATABASE UniversityDB;  USE UniversityDB;  -- Create Student Table  CREATE TABLE Student (  snum INT PRIMARY KEY,  sname VARCHAR(100),  major VARCHAR(50),  level VARCHAR(2), -- Levels: FR, SO, JR, SR  age INT  );  -- Create Class Table  CREATE TABLE Class (  name VARCHAR(100) PRIMARY KEY,  meets\_at VARCHAR(50),  room VARCHAR(50),  fid INT  );  -- Create Enrolled Table  CREATE TABLE Enrolled (  snum INT,  cname VARCHAR(100),  PRIMARY KEY (snum, cname),  FOREIGN KEY (snum) REFERENCES Student(snum),  FOREIGN KEY (cname) REFERENCES Class(name)  );  -- Create Faculty Table  CREATE TABLE Faculty (  fid INT PRIMARY KEY,  fname VARCHAR(100),  deptid INT  );  -- Insert Sample Data for Testing  INSERT INTO Student (snum, sname, major, level, age) VALUES  (1, 'Alice', 'History', 'JR', 20),  (2, 'Bob', 'Math', 'SR', 22),  (3, 'Charlie', 'History', 'SO', 19),  (4, 'Diana', 'Physics', 'FR', 18),  (5, 'Eve', 'CS', 'JR', 21);  INSERT INTO Faculty (fid, fname, deptid) VALUES  (1, 'I. Teach', 10),  (2, 'Dr. Smith', 20);  INSERT INTO Class (name, meets\_at, room, fid) VALUES  ('Math101', '10:00AM', 'R128', 1),  ('Hist202', '11:00AM', 'R128', 1),  ('Phys303', '10:00AM', 'R129', 2);  INSERT INTO Enrolled (snum, cname) VALUES  (1, 'Hist202'),  (2, 'Math101'),  (3, 'Hist202'),  (4, 'Phys303'),  (5, 'Math101');  -- Step 2: Queries  -- 1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach.  SELECT DISTINCT s.sname  FROM Student s  JOIN Enrolled e ON s.snum = e.snum  JOIN Class c ON e.cname = c.name  JOIN Faculty f ON c.fid = f.fid  WHERE s.level = 'JR' AND f.fname = 'I. Teach';  -- 2. Find the age of the oldest student who is either a History major or enrolled in a course taught by I. Teach.  SELECT MAX(s.age) AS OldestAge  FROM Student s  LEFT JOIN Enrolled e ON s.snum = e.snum  LEFT JOIN Class c ON e.cname = c.name  LEFT JOIN Faculty f ON c.fid = f.fid  WHERE s.major = 'History' OR f.fname = 'I. Teach';  -- 3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.  SELECT DISTINCT c.name  FROM Class c  LEFT JOIN Enrolled e ON c.name = e.cname  GROUP BY c.name  HAVING c.room = 'R128' OR COUNT(e.snum) >= 5;  -- 4. Find the names of all students who are enrolled in two classes that meet at the same time.  SELECT DISTINCT s.sname  FROM Student s  JOIN Enrolled e1 ON s.snum = e1.snum  JOIN Enrolled e2 ON s.snum = e2.snum  JOIN Class c1 ON e1.cname = c1.name  JOIN Class c2 ON e2.cname = c2.name  WHERE c1.meets\_at = c2.meets\_at AND e1.cname <> e2.cname;  -- 5. Find the names of faculty members who teach in every room in which some class is taught.  SELECT DISTINCT f.fname  FROM Faculty f  JOIN Class c ON f.fid = c.fid  WHERE NOT EXISTS (  SELECT DISTINCT room  FROM Class c2  WHERE c2.room NOT IN (  SELECT room  FROM Class c3  WHERE c3.fid = f.fid  )  );  -- 6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.  SELECT DISTINCT f.fname  FROM Faculty f  LEFT JOIN Class c ON f.fid = c.fid  LEFT JOIN Enrolled e ON c.name = e.cname  GROUP BY f.fid  HAVING COUNT(e.snum) < 5;  -- 7. For each level, print the level and the average age of students for that level.  SELECT s.level, AVG(s.age) AS AvgAge  FROM Student s  GROUP BY s.level;  -- 8. For all levels except JR, print the level and the average age of students for that level.  SELECT s.level, AVG(s.age) AS AvgAge  FROM Student s  WHERE s.level <> 'JR'  GROUP BY s.level;  -- 9. For each faculty member that has taught classes only in room R128, print the faculty member’s name and the total number of classes they have taught.  SELECT f.fname, COUNT(c.name) AS TotalClasses  FROM Faculty f  JOIN Class c ON f.fid = c.fid  GROUP BY f.fid  HAVING COUNT(DISTINCT c.room) = 1 AND MAX(c.room) = 'R128';  -- 10. Find the names of students enrolled in the maximum number of classes.  SELECT s.sname  FROM Student s  JOIN Enrolled e ON s.snum = e.snum  GROUP BY s.snum  HAVING COUNT(e.cname) = (  SELECT MAX(class\_count)  FROM (  SELECT COUNT(e1.cname) AS class\_count  FROM Enrolled e1  GROUP BY e1.snum  ) subquery  );  -- 11. Find the names of students not enrolled in any class.  SELECT s.sname  FROM Student s  LEFT JOIN Enrolled e ON s.snum = e.snum  WHERE e.snum IS NULL;  -- 12. For each age value that appears in Students, find the level value that appears most often.  SELECT age, level  FROM (  SELECT age, level, COUNT(\*) AS level\_count,  RANK() OVER (PARTITION BY age ORDER BY COUNT(\*) DESC) AS rnk  FROM Student  GROUP BY age, level  ) subquery  WHERE rnk = 1; |
| 13 | Consider the following schema:  Suppliers(sid: integer, sname: string, address: string)  Parts(pid: integer, pname: string, color: string)  Catalog(sid: integer, pid: integer, cost: real)  The Catalog relation lists the prices charged for parts by Suppliers. Write the following  queries in SQL:  1. Find the pnames of parts for which there is some supplier.  2. Find the snames of suppliers who supply every part.  3. Find the snames of suppliers who supply every red part.  4. Find the pnames of parts supplied by Acme Widget Suppliers and no one else.  5. Find the sids of suppliers who charge more for some part than the average cost of  that part (averaged over all the suppliers who supply that part).  6. For each part, find the sname of the supplier who charges the most for that part.  7. Find the sids of suppliers who supply only red parts.  8. Find the sids of suppliers who supply a red part and a green part.  9. Find the sids of suppliers who supply a red part or a green part.  10. For every supplier that only supplies green parts, print the name of the supplier  and the total number of parts that she supplies.  11. For every supplier that supplies a green part and a red part, print the name and  price of the most expensive part that she supplies. |
|  | -- Step 1: Create Database and Tables  CREATE DATABASE SupplyChainDB;  USE SupplyChainDB;  -- Create Suppliers Table  CREATE TABLE Suppliers (  sid INT PRIMARY KEY,  sname VARCHAR(100),  address VARCHAR(255)  );  -- Create Parts Table  CREATE TABLE Parts (  pid INT PRIMARY KEY,  pname VARCHAR(100),  color VARCHAR(50)  );  -- Create Catalog Table  CREATE TABLE Catalog (  sid INT,  pid INT,  cost DECIMAL(10, 2),  PRIMARY KEY (sid, pid),  FOREIGN KEY (sid) REFERENCES Suppliers(sid),  FOREIGN KEY (pid) REFERENCES Parts(pid)  );  -- Insert Sample Data  INSERT INTO Suppliers (sid, sname, address) VALUES  (1, 'Acme Widget Suppliers', '123 Main St'),  (2, 'Best Parts Co', '456 Elm St'),  (3, 'Quality Supplies', '789 Oak St');  INSERT INTO Parts (pid, pname, color) VALUES  (101, 'Bolt', 'Red'),  (102, 'Nut', 'Green'),  (103, 'Screw', 'Blue'),  (104, 'Washer', 'Red');  INSERT INTO Catalog (sid, pid, cost) VALUES  (1, 101, 5.00),  (1, 102, 3.00),  (2, 101, 6.00),  (2, 103, 2.50),  (3, 102, 2.75),  (3, 104, 4.00);  -- Step 2: Queries  -- 1. Find the pnames of parts for which there is some supplier.  SELECT DISTINCT p.pname  FROM Parts p  JOIN Catalog c ON p.pid = c.pid;  -- 2. Find the snames of suppliers who supply every part.  SELECT s.sname  FROM Suppliers s  WHERE NOT EXISTS (  SELECT p.pid  FROM Parts p  WHERE NOT EXISTS (  SELECT c.pid  FROM Catalog c  WHERE c.sid = s.sid AND c.pid = p.pid  )  );  -- 3. Find the snames of suppliers who supply every red part.  SELECT s.sname  FROM Suppliers s  WHERE NOT EXISTS (  SELECT p.pid  FROM Parts p  WHERE p.color = 'Red'  AND NOT EXISTS (  SELECT c.pid  FROM Catalog c  WHERE c.sid = s.sid AND c.pid = p.pid  )  );  -- 4. Find the pnames of parts supplied by Acme Widget Suppliers and no one else.  SELECT p.pname  FROM Parts p  JOIN Catalog c1 ON p.pid = c1.pid  JOIN Suppliers s ON c1.sid = s.sid  WHERE s.sname = 'Acme Widget Suppliers'  AND NOT EXISTS (  SELECT \*  FROM Catalog c2  WHERE c2.pid = c1.pid AND c2.sid <> c1.sid  );  -- 5. Find the sids of suppliers who charge more for some part than the average cost of that part.  SELECT DISTINCT c.sid  FROM Catalog c  JOIN (  SELECT pid, AVG(cost) AS avg\_cost  FROM Catalog  GROUP BY pid  ) avg\_costs ON c.pid = avg\_costs.pid  WHERE c.cost > avg\_costs.avg\_cost;  -- 6. For each part, find the sname of the supplier who charges the most for that part.  SELECT p.pname, s.sname  FROM Parts p  JOIN Catalog c ON p.pid = c.pid  JOIN Suppliers s ON c.sid = s.sid  WHERE c.cost = (  SELECT MAX(c1.cost)  FROM Catalog c1  WHERE c1.pid = p.pid  );  -- 7. Find the sids of suppliers who supply only red parts.  SELECT s.sid  FROM Suppliers s  WHERE NOT EXISTS (  SELECT c.pid  FROM Catalog c  JOIN Parts p ON c.pid = p.pid  WHERE c.sid = s.sid AND p.color <> 'Red'  );  -- 8. Find the sids of suppliers who supply a red part and a green part.  SELECT DISTINCT c1.sid  FROM Catalog c1  JOIN Parts p1 ON c1.pid = p1.pid  JOIN Catalog c2 ON c1.sid = c2.sid  JOIN Parts p2 ON c2.pid = p2.pid  WHERE p1.color = 'Red' AND p2.color = 'Green';  -- 9. Find the sids of suppliers who supply a red part or a green part.  SELECT DISTINCT c.sid  FROM Catalog c  JOIN Parts p ON c.pid = p.pid  WHERE p.color = 'Red' OR p.color = 'Green';  -- 10. For every supplier that only supplies green parts, print the name of the supplier and the total number of parts that they supply.  SELECT s.sname, COUNT(c.pid) AS TotalParts  FROM Suppliers s  JOIN Catalog c ON s.sid = c.sid  JOIN Parts p ON c.pid = p.pid  WHERE NOT EXISTS (  SELECT c1.pid  FROM Catalog c1  JOIN Parts p1 ON c1.pid = p1.pid  WHERE c1.sid = s.sid AND p1.color <> 'Green'  )  GROUP BY s.sid;  -- 11. For every supplier that supplies a green part and a red part, print the name and price of the most expensive part that they supply.  SELECT s.sname, MAX(c.cost) AS MostExpensivePartCost  FROM Suppliers s  JOIN Catalog c ON s.sid = c.sid  JOIN Parts p1 ON c.pid = p1.pid  WHERE EXISTS (  SELECT c1.sid  FROM Catalog c1  JOIN Parts p2 ON c1.pid = p2.pid  WHERE c1.sid = s.sid AND p2.color = 'Red'  )  AND EXISTS (  SELECT c2.sid  FROM Catalog c2  JOIN Parts p3 ON c2.pid = p3.pid  WHERE c2.sid = s.sid AND p3.color = 'Green'  )  GROUP BY s.sid; |
| 14 | The following relations keep track of airline flight information:  Flights(flno: integer, from: string, to: string, distance: integer,  departs: time, arrives: time, price: real)  Aircraft(aid: integer, aname: string, cruisingrange: integer)  Certified(eid: integer, aid: integer)  Employees(eid: integer, ename: string, salary: integer)  Note that the Employees relation describes pilots and other kinds of employees as well;  every pilot is certified for some aircraft, and only pilots are certified to fly. Write each  of the following queries in SQL.  1. Find the names of aircraft such that all pilots certified to operate them have  salaries more than $80,000.  2. For each pilot who is certified for more than three aircraft, find the eid and the  maximum cruisingrange of the aircraft for which she or he is certified.  3. Find the names of pilots whose salary is less than the price of the cheapest route  from Los Angeles to Honolulu.  4. For all aircraft with cruisingrange over 1000 miles, find the name of the aircraft  and the average salary of all pilots certified for this aircraft.  5. Find the names of pilots certified for some Boeing aircraft.  6. Find the aids of all aircraft that can be used on routes from Los Angeles to  Chicago.  7. Identify the routes that can be piloted by every pilot who makes more than  $100,000.  8. Print the enames of pilots who can operate planes with cruisingrange greater than 3000 miles but are not certified on any Boeing aircraft.  9. A customer wants to travel from Madison to New York with no more than two  changes of flight. List the choice of departure times from Madison if the customer  wants to arrive in New York by 6 p.m.  10. Compute the difference between the average salary of a pilot and the average  salary of all employees (including pilots).  11. Print the name and salary of every nonpilot whose salary is more than the average  salary for pilots.  12. Print the names of employees who are certified only on aircrafts with cruising  range longer than 1000 miles.  13. Print the names of employees who are certified only on aircrafts with cruising  range longer than 1000 miles, but on at least two such aircrafts.  14. Print the names of employees who are certified only on aircrafts with cruising  range longer than 1000 miles and who are certified on some Boeing aircraft. |
|  | -- Step 1: Create Database and Tables  CREATE DATABASE AirlineDB;  USE AirlineDB;  -- Create Flights Table  CREATE TABLE Flights (  flno INT PRIMARY KEY,  `from` VARCHAR(100),  `to` VARCHAR(100),  distance INT,  departs TIME,  arrives TIME,  price DECIMAL(10, 2)  );  -- Create Aircraft Table  CREATE TABLE Aircraft (  aid INT PRIMARY KEY,  aname VARCHAR(100),  cruisingrange INT  );  -- Create Certified Table  CREATE TABLE Certified (  eid INT,  aid INT,  PRIMARY KEY (eid, aid),  FOREIGN KEY (aid) REFERENCES Aircraft(aid)  );  -- Create Employees Table  CREATE TABLE Employees (  eid INT PRIMARY KEY,  ename VARCHAR(100),  salary INT  );  -- Insert Sample Data  INSERT INTO Flights (flno, `from`, `to`, distance, departs, arrives, price) VALUES  (1, 'Los Angeles', 'Honolulu', 2500, '09:00:00', '13:00:00', 300.00),  (2, 'Los Angeles', 'Chicago', 2000, '10:00:00', '14:00:00', 200.00),  (3, 'Madison', 'New York', 1500, '07:00:00', '10:00:00', 150.00);  INSERT INTO Aircraft (aid, aname, cruisingrange) VALUES  (101, 'Boeing 747', 3500),  (102, 'Airbus A320', 3000),  (103, 'Cessna 172', 500);  INSERT INTO Employees (eid, ename, salary) VALUES  (1, 'John Doe', 90000),  (2, 'Jane Smith', 95000),  (3, 'Alice Brown', 75000),  (4, 'Bob Johnson', 85000),  (5, 'Mark Davis', 40000);  INSERT INTO Certified (eid, aid) VALUES  (1, 101),  (1, 102),  (2, 101),  (2, 103),  (3, 102),  (4, 101),  (5, 103);  -- Step 2: Queries  -- 1. Find the names of aircraft such that all pilots certified to operate them have salaries more than $80,000.  SELECT a.aname  FROM Aircraft a  WHERE NOT EXISTS (  SELECT c.eid  FROM Certified c  JOIN Employees e ON c.eid = e.eid  WHERE c.aid = a.aid AND e.salary <= 80000  );  -- 2. For each pilot who is certified for more than three aircraft, find the eid and the maximum cruisingrange of the aircraft for which she or he is certified.  SELECT c.eid, MAX(a.cruisingrange) AS MaxCruisingRange  FROM Certified c  JOIN Aircraft a ON c.aid = a.aid  GROUP BY c.eid  HAVING COUNT(c.aid) > 3;  -- 3. Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.  SELECT e.ename  FROM Employees e  WHERE e.salary < (  SELECT MIN(f.price)  FROM Flights f  WHERE f.from = 'Los Angeles' AND f.to = 'Honolulu'  );  -- 4. For all aircraft with cruisingrange over 1000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.  SELECT a.aname, AVG(e.salary) AS AvgSalary  FROM Aircraft a  JOIN Certified c ON a.aid = c.aid  JOIN Employees e ON c.eid = e.eid  WHERE a.cruisingrange > 1000  GROUP BY a.aname;  -- 5. Find the names of pilots certified for some Boeing aircraft.  SELECT DISTINCT e.ename  FROM Employees e  JOIN Certified c ON e.eid = c.eid  JOIN Aircraft a ON c.aid = a.aid  WHERE a.aname LIKE 'Boeing%';  -- 6. Find the aids of all aircraft that can be used on routes from Los Angeles to Chicago.  SELECT DISTINCT a.aid  FROM Aircraft a  JOIN Flights f ON f.distance <= a.cruisingrange  WHERE f.from = 'Los Angeles' AND f.to = 'Chicago';  -- 7. Identify the routes that can be piloted by every pilot who makes more than $100,000.  SELECT f.flno, f.from, f.to  FROM Flights f  WHERE NOT EXISTS (  SELECT e.eid  FROM Employees e  WHERE e.salary > 100000  AND NOT EXISTS (  SELECT c.aid  FROM Certified c  JOIN Aircraft a ON c.aid = a.aid  WHERE a.cruisingrange >= f.distance AND c.eid = e.eid  )  );  -- 8. Print the enames of pilots who can operate planes with cruisingrange greater than 3000 miles but are not certified on any Boeing aircraft.  SELECT DISTINCT e.ename  FROM Employees e  JOIN Certified c ON e.eid = c.eid  JOIN Aircraft a ON c.aid = a.aid  WHERE a.cruisingrange > 3000  AND e.eid NOT IN (  SELECT c1.eid  FROM Certified c1  JOIN Aircraft a1 ON c1.aid = a1.aid  WHERE a1.aname LIKE 'Boeing%'  );  -- 9. A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.  SELECT DISTINCT f1.departs AS DepartureTime  FROM Flights f1  JOIN Flights f2 ON f1.to = f2.from  WHERE f1.from = 'Madison' AND f2.to = 'New York'  AND TIMEDIFF('18:00:00', f2.arrives) >= 0;  -- 10. Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).  SELECT (  SELECT AVG(e1.salary) FROM Employees e1  JOIN Certified c ON e1.eid = c.eid  ) - AVG(e2.salary) AS SalaryDifference  FROM Employees e2;  -- 11. Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.  SELECT e.ename, e.salary  FROM Employees e  WHERE e.eid NOT IN (SELECT DISTINCT c.eid FROM Certified c)  AND e.salary > (  SELECT AVG(e1.salary)  FROM Employees e1  JOIN Certified c ON e1.eid = c.eid  );  -- 12. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.  SELECT DISTINCT e.ename  FROM Employees e  WHERE NOT EXISTS (  SELECT c.aid  FROM Certified c  JOIN Aircraft a ON c.aid = a.aid  WHERE c.eid = e.eid AND a.cruisingrange <= 1000  );  -- 13. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts.  SELECT e.ename  FROM Employees e  JOIN Certified c ON e.eid = c.eid  JOIN Aircraft a ON c.aid = a.aid  WHERE a.cruisingrange > 1000  GROUP BY e.eid  HAVING COUNT(DISTINCT c.aid) >= 2;  -- 14. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft.  SELECT DISTINCT e.ename  FROM Employees e  JOIN Certified c ON e.eid = c.eid  JOIN Aircraft a ON c.aid = a.aid  WHERE a.cruisingrange > 1000  AND EXISTS (  SELECT 1  FROM Certified c1  JOIN Aircraft a1 ON c1.aid = a1.aid  WHERE c1.eid = e.eid AND a1.aname LIKE 'Boeing%'  ); |
| 15 | Consider the following relational schema and briefly answer the questions that follow:  Emp(eid: integer, ename: string, age: integer, salary: real)  Works(eid: integer, did: integer, pct time: integer)  Dept(did: integer, budget: real, managerid: integer)  1. Define a table constraint on Emp that will ensure that every employee makes at  least $10,000.  2. Define a table constraint on Dept that will ensure that all managers have age > 30.  3. Define an assertion on Dept that will ensure that all managers have age > 30.  Compare this assertion with the equivalent table constraint. Explain which is better.  4. Write SQL statements to delete all information about employees whose salaries  exceed that of the manager of one or more departments that they work in. Be sure to ensure that all the relevant integrity constraints are satisfied after your updates. |
|  | -- Drop existing tables if they exist  DROP TABLE IF EXISTS Works;  DROP TABLE IF EXISTS Dept;  DROP TABLE IF EXISTS Emp;  -- 1. Create Emp table with a CHECK constraint for salary  CREATE TABLE Emp (  eid INT PRIMARY KEY,  ename VARCHAR(100),  age INT,  salary REAL CHECK (salary >= 10000)  );  -- 2. Create Dept table with a CHECK constraint for manager age > 30  CREATE TABLE Dept (  did INT PRIMARY KEY,  budget REAL,  managerid INT,  FOREIGN KEY (managerid) REFERENCES Emp(eid)  );  -- Trigger to enforce manager age > 30  DELIMITER //  CREATE TRIGGER chk\_manager\_age BEFORE INSERT ON Dept  FOR EACH ROW  BEGIN  DECLARE manager\_age INT;  SELECT age INTO manager\_age FROM Emp WHERE eid = NEW.managerid;  IF manager\_age <= 30 THEN  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT = 'Manager age must be greater than 30';  END IF;  END;  //  DELIMITER ;  -- 3. Create Works table  CREATE TABLE Works (  eid INT,  did INT,  pct\_time INT,  FOREIGN KEY (eid) REFERENCES Emp(eid),  FOREIGN KEY (did) REFERENCES Dept(did)  );  -- Insert sample data  INSERT INTO Emp (eid, ename, age, salary) VALUES  (1, 'John', 45, 12000),  (2, 'Jane', 50, 25000),  (3, 'Alice', 28, 9000), -- Will fail due to salary constraint  (4, 'Bob', 40, 15000);  INSERT INTO Dept (did, budget, managerid) VALUES  (101, 500000, 1),  (102, 300000, 2);  INSERT INTO Works (eid, did, pct\_time) VALUES  (1, 101, 50),  (2, 102, 75),  (4, 101, 25);  -- 4. Delete employees whose salary exceeds their manager's salary  DELETE FROM Emp  WHERE eid IN (  SELECT e.eid  FROM Emp e  JOIN Works w ON e.eid = w.eid  JOIN Dept d ON w.did = d.did  JOIN Emp m ON d.managerid = m.eid  WHERE e.salary > m.salary  );  -- Test Queries  -- Ensure the structure and constraints are correctly implemented  SELECT \* FROM Emp;  SELECT \* FROM Dept;  SELECT \* FROM Works; |
| 16 | Consider the following relational schema. An employee can work in more than one department; the pct time field of the Works relation shows the percentage of time that a given employee works in a given department.  Emp(eid: integer, ename: string, age: integer, salary: real)  Works(eid: integer, did: integer, pct time: integer)  Dept(did: integer, budget: real, managerid: integer)  Write SQL-92 integrity constraints (domain, key, foreign key, or CHECK constraints; or  assertions) or SQL:1999 triggers to ensure each of the following requirements, consid-  ered independently.  1. Employees must make a minimum salary of $1000.  2. Every manager must be also be an employee.  3. The total percentage of all appointments for an employee must be under 100%.  4. A manager must always have a higher salary than any employee that he or she  manages.  5. Whenever an employee is given a raise, the manager’s salary must be increased to  be at least as much.  6. Whenever an employee is given a raise, the manager’s salary must be increased  to be at least as much. Further, whenever an employee is given a raise, the  department’s budget must be increased to be greater than the sum of salaries of  all employees in the department. |
|  | -- Drop existing tables if they exist  DROP TABLE IF EXISTS Works;  DROP TABLE IF EXISTS Dept;  DROP TABLE IF EXISTS Emp;  -- 1. Create Emp table with a CHECK constraint for minimum salary  CREATE TABLE Emp (  eid INT PRIMARY KEY,  ename VARCHAR(100),  age INT,  salary REAL CHECK (salary >= 1000)  );  -- 2. Create Dept table with a FOREIGN KEY constraint for manager (must be an employee)  CREATE TABLE Dept (  did INT PRIMARY KEY,  budget REAL,  managerid INT,  FOREIGN KEY (managerid) REFERENCES Emp(eid)  );  -- 3. Create Works table (employees working in multiple departments with pct\_time)  CREATE TABLE Works (  eid INT,  did INT,  pct\_time INT,  PRIMARY KEY (eid, did),  FOREIGN KEY (eid) REFERENCES Emp(eid),  FOREIGN KEY (did) REFERENCES Dept(did)  );  -- 4. Trigger to ensure the total pct\_time for an employee in all departments does not exceed 100%  DELIMITER //  CREATE TRIGGER check\_pct\_time BEFORE INSERT ON Works  FOR EACH ROW  BEGIN  DECLARE total\_pct INT;  -- Sum pct\_time for the employee in all departments  SELECT SUM(pct\_time) INTO total\_pct  FROM Works  WHERE eid = NEW.eid;  IF (total\_pct + NEW.pct\_time) > 100 THEN  SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Total pct\_time for an employee cannot exceed 100%';  END IF;  END;  //  DELIMITER ;  -- 5. Trigger to ensure that a manager's salary is higher than those of employees they manage  DELIMITER //  CREATE TRIGGER check\_manager\_salary BEFORE INSERT ON Dept  FOR EACH ROW  BEGIN  DECLARE manager\_salary REAL;  DECLARE employee\_salary REAL;  -- Get the salary of the manager  SELECT salary INTO manager\_salary FROM Emp WHERE eid = NEW.managerid;  -- Get the salary of employees working under this manager  DECLARE cur CURSOR FOR  SELECT e.salary  FROM Emp e  JOIN Works w ON e.eid = w.eid  WHERE w.did = NEW.did;    OPEN cur;  FETCH cur INTO employee\_salary;  WHILE (employee\_salary IS NOT NULL) DO  IF employee\_salary >= manager\_salary THEN  SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Manager salary must be greater than employees under them';  END IF;  FETCH cur INTO employee\_salary;  END WHILE;  CLOSE cur;  END;  //  DELIMITER ;  -- 6. Trigger to ensure the manager's salary is increased if an employee receives a raise  DELIMITER //  CREATE TRIGGER raise\_manager\_salary BEFORE UPDATE ON Emp  FOR EACH ROW  BEGIN  DECLARE manager\_salary REAL;  -- Get the manager's salary  SELECT salary INTO manager\_salary FROM Emp WHERE eid = (  SELECT managerid FROM Dept WHERE managerid = NEW.eid  );  -- If the new salary is greater than the manager's salary, raise the manager's salary  IF NEW.salary > manager\_salary THEN  UPDATE Emp  SET salary = NEW.salary  WHERE eid = (  SELECT managerid FROM Dept WHERE managerid = NEW.eid  );  END IF;  END;  //  DELIMITER ;  -- 7. Trigger to ensure the manager’s salary is increased if an employee is given a raise and department budget is updated  DELIMITER //  CREATE TRIGGER raise\_manager\_and\_budget BEFORE UPDATE ON Emp  FOR EACH ROW  BEGIN  DECLARE manager\_salary REAL;  DECLARE department\_budget REAL;  -- Get the manager's salary  SELECT salary INTO manager\_salary FROM Emp WHERE eid = (  SELECT managerid FROM Dept WHERE managerid = NEW.eid  );  -- Ensure manager salary is increased if employee salary is increased  IF NEW.salary > manager\_salary THEN  UPDATE Emp  SET salary = NEW.salary  WHERE eid = (  SELECT managerid FROM Dept WHERE managerid = NEW.eid  );  END IF;  -- Update the department budget to be greater than the sum of salaries of all employees in the department  SELECT SUM(salary) INTO department\_budget  FROM Emp  JOIN Works w ON Emp.eid = w.eid  WHERE w.did = (  SELECT did FROM Works WHERE eid = NEW.eid  );  -- Update the department budget  UPDATE Dept  SET budget = department\_budget + 5000 -- Example increase  WHERE did = (  SELECT did FROM Works WHERE eid = NEW.eid  );  END;  //  DELIMITER ;  -- Test Queries  -- Insert sample data  INSERT INTO Emp (eid, ename, age, salary) VALUES  (1, 'John', 45, 12000),  (2, 'Jane', 50, 25000),  (3, 'Alice', 28, 9000), -- Will fail due to salary constraint  (4, 'Bob', 40, 15000);  INSERT INTO Dept (did, budget, managerid) VALUES  (101, 500000, 1),  (102, 300000, 2);  INSERT INTO Works (eid, did, pct\_time) VALUES  (1, 101, 50),  (2, 102, 75),  (4, 101, 25);  -- Test integrity constraints  SELECT \* FROM Emp;  SELECT \* FROM Dept;  SELECT \* FROM Works; |
| 17 | Implement nested sub queries. Perform a test for set membership (in, not in), set comparison (<some, >=some, <all etc.) and set cardinality (unique, not unique).  Assume suitable tables. |
|  | -- Drop existing tables if they exist  DROP TABLE IF EXISTS Orders;  DROP TABLE IF EXISTS Customers;  DROP TABLE IF EXISTS Products;  DROP TABLE IF EXISTS Employees;  -- Create Customers table  CREATE TABLE Customers (  customer\_id INT PRIMARY KEY,  customer\_name VARCHAR(100),  city VARCHAR(50),  age INT  );  -- Create Employees table  CREATE TABLE Employees (  employee\_id INT PRIMARY KEY,  employee\_name VARCHAR(100),  department\_id INT,  salary REAL  );  -- Create Products table  CREATE TABLE Products (  product\_id INT PRIMARY KEY,  product\_name VARCHAR(100),  price REAL  );  -- Create Orders table  CREATE TABLE Orders (  order\_id INT PRIMARY KEY,  customer\_id INT,  product\_id INT,  order\_date DATE,  quantity INT,  FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id),  FOREIGN KEY (product\_id) REFERENCES Products(product\_id)  );  -- Insert sample data into Customers table  INSERT INTO Customers (customer\_id, customer\_name, city, age) VALUES  (1, 'Alice', 'New York', 30),  (2, 'Bob', 'Los Angeles', 45),  (3, 'Charlie', 'Chicago', 35),  (4, 'David', 'Houston', 50);  -- Insert sample data into Employees table  INSERT INTO Employees (employee\_id, employee\_name, department\_id, salary) VALUES  (1, 'John', 101, 55000),  (2, 'Jane', 102, 65000),  (3, 'Mark', 101, 45000),  (4, 'Lucy', 103, 70000);  -- Insert sample data into Products table  INSERT INTO Products (product\_id, product\_name, price) VALUES  (1, 'Laptop', 1000),  (2, 'Smartphone', 500),  (3, 'Tablet', 300),  (4, 'Monitor', 250);  -- Insert sample data into Orders table  INSERT INTO Orders (order\_id, customer\_id, product\_id, order\_date, quantity) VALUES  (1, 1, 1, '2024-11-01', 2),  (2, 1, 2, '2024-11-02', 3),  (3, 2, 3, '2024-11-03', 5),  (4, 3, 1, '2024-11-04', 1),  (5, 4, 4, '2024-11-05', 6);  -- \*\*Set Membership: IN\*\*  -- 1. Find all customers who have placed orders for either 'Laptop' or 'Smartphone'  SELECT customer\_name  FROM Customers  WHERE customer\_id IN (  SELECT DISTINCT customer\_id  FROM Orders  WHERE product\_id IN (  SELECT product\_id FROM Products WHERE product\_name IN ('Laptop', 'Smartphone')  )  );  -- \*\*Set Membership: NOT IN\*\*  -- 2. Find all employees whose salary is not equal to the average salary of the department they work in  SELECT employee\_name  FROM Employees  WHERE employee\_id NOT IN (  SELECT employee\_id  FROM Employees  WHERE salary = (  SELECT AVG(salary)  FROM Employees  WHERE department\_id = Employees.department\_id  )  );  -- \*\*Set Comparison: < ALL\*\*  -- 3. Find all customers whose total quantity ordered is less than the quantity ordered for 'Laptop' by all other customers  SELECT customer\_name  FROM Customers  WHERE customer\_id IN (  SELECT customer\_id  FROM Orders  WHERE quantity < ALL (  SELECT quantity  FROM Orders  WHERE product\_id = (SELECT product\_id FROM Products WHERE product\_name = 'Laptop')  )  );  -- \*\*Set Comparison: > ANY\*\*  -- 4. Find all employees who have a salary greater than at least one employee in department 101  SELECT employee\_name  FROM Employees  WHERE salary > ANY (  SELECT salary  FROM Employees  WHERE department\_id = 101  );  -- \*\*Set Comparison: >= ALL\*\*  -- 5. Find all products that are priced greater than or equal to the price of every other product  SELECT product\_name  FROM Products  WHERE price >= ALL (  SELECT price  FROM Products  WHERE product\_id != Products.product\_id  );  -- \*\*Set Cardinality: UNIQUE\*\*  -- 6. Find all customers who have placed unique orders (not repeated product orders)  SELECT customer\_name  FROM Customers  WHERE customer\_id IN (  SELECT customer\_id  FROM Orders  GROUP BY customer\_id, product\_id  HAVING COUNT(\*) = 1  );  -- \*\*Set Cardinality: NOT UNIQUE\*\*  -- 7. Find all products that have been ordered by multiple customers  SELECT product\_name  FROM Products  WHERE product\_id IN (  SELECT product\_id  FROM Orders  GROUP BY product\_id  HAVING COUNT(DISTINCT customer\_id) > 1  );  -- \*\*Set Comparison: < SOME\*\*  -- 8. Find all customers who have ordered fewer products than at least one customer who ordered 'Smartphone'  SELECT customer\_name  FROM Customers  WHERE customer\_id IN (  SELECT customer\_id  FROM Orders  WHERE quantity < SOME (  SELECT quantity  FROM Orders  WHERE product\_id = (SELECT product\_id FROM Products WHERE product\_name = 'Smartphone')  )  );  -- \*\*Set Membership: IN (with nested query)\*\*  -- 9. Find all employees who work in departments where the average salary is above $60,000  SELECT employee\_name  FROM Employees  WHERE department\_id IN (  SELECT department\_id  FROM Employees  GROUP BY department\_id  HAVING AVG(salary) > 60000  );  -- \*\*Set Comparison: > ALL\*\*  -- 10. Find all employees whose salary is greater than the salaries of all employees in department 103  SELECT employee\_name  FROM Employees  WHERE salary > ALL (  SELECT salary  FROM Employees  WHERE department\_id = 103  ); |
| 18 | Execute DDL statements which demonstrate the use of views. Try to update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables. |
|  | -- Drop existing tables if they exist  DROP TABLE IF EXISTS Employees;  DROP TABLE IF EXISTS Departments;  DROP TABLE IF EXISTS Employees\_View;  -- Create Employees table  CREATE TABLE Employees (  employee\_id INT PRIMARY KEY,  employee\_name VARCHAR(100),  department\_id INT,  salary REAL,  hire\_date DATE  );  -- Create Departments table  CREATE TABLE Departments (  department\_id INT PRIMARY KEY,  department\_name VARCHAR(100),  manager\_id INT  );  -- Insert sample data into Employees table  INSERT INTO Employees (employee\_id, employee\_name, department\_id, salary, hire\_date) VALUES  (1, 'John Doe', 101, 50000, '2020-05-15'),  (2, 'Jane Smith', 102, 60000, '2019-07-30'),  (3, 'Mark Johnson', 101, 55000, '2018-03-20'),  (4, 'Lucy Brown', 103, 70000, '2021-08-10'),  (5, 'Charlie Black', 102, 65000, '2022-01-25');  -- Insert sample data into Departments table  INSERT INTO Departments (department\_id, department\_name, manager\_id) VALUES  (101, 'Sales', 1),  (102, 'Marketing', 2),  (103, 'Engineering', 4);  -- 1. Create a simple view to show employee names and their salaries  CREATE VIEW Employees\_View AS  SELECT employee\_name, salary  FROM Employees;  -- Query the view  SELECT \* FROM Employees\_View;  -- 2. Try updating the base table using the view (this should be allowed as it's simple and based on a single table)  UPDATE Employees\_View  SET salary = salary + 5000  WHERE employee\_name = 'John Doe';  -- Verify the update on the base table (Employees)  SELECT \* FROM Employees WHERE employee\_name = 'John Doe';  -- 3. Create a view with multiple tables by joining Employees and Departments  CREATE VIEW Employees\_Departments\_View AS  SELECT e.employee\_name, e.salary, d.department\_name  FROM Employees e  JOIN Departments d ON e.department\_id = d.department\_id;  -- Query the view that combines data from Employees and Departments  SELECT \* FROM Employees\_Departments\_View;  -- 4. Try updating the base tables using the combined view (this should have restrictions)  -- Here, we will attempt to update a field from multiple tables, which is not allowed with this type of view.  -- Attempt to update the salary of an employee (this works as it is directly related to the Employees table)  UPDATE Employees\_Departments\_View  SET salary = salary + 1000  WHERE employee\_name = 'Jane Smith';  -- Verify the update on the base table (Employees)  SELECT \* FROM Employees WHERE employee\_name = 'Jane Smith';  -- Attempt to update the department name (this won't work as it's from the Departments table)  -- This will cause an error because updating a column from a joined view is not allowed unless explicitly handled.  UPDATE Employees\_Departments\_View  SET department\_name = 'Product Management'  WHERE employee\_name = 'Mark Johnson';  -- 5. Create a view with a WHERE clause that can filter data based on conditions (salary above 60000)  CREATE VIEW HighSalaryEmployees\_View AS  SELECT employee\_name, salary  FROM Employees  WHERE salary > 60000;  -- Query the view with the condition applied  SELECT \* FROM HighSalaryEmployees\_View;  -- 6. Check if the view is updatable: If the view is based on a single table and doesn't involve JOINs or GROUP BY, it will be updatable.  -- For this, let's check if we can update the Employees table via the HighSalaryEmployees\_View.  UPDATE HighSalaryEmployees\_View  SET salary = salary + 2000  WHERE employee\_name = 'Jane Smith';  -- Verify the update on the base table (Employees)  SELECT \* FROM Employees WHERE employee\_name = 'Jane Smith';  -- 7. Create an aggregate view (this view will not be updatable)  CREATE VIEW DepartmentSalaryStats AS  SELECT d.department\_name, COUNT(e.employee\_id) AS num\_employees, AVG(e.salary) AS avg\_salary  FROM Employees e  JOIN Departments d ON e.department\_id = d.department\_id  GROUP BY d.department\_name;  -- Query the aggregate view  SELECT \* FROM DepartmentSalaryStats;  -- Attempt to update the aggregate view (this will not work because it's based on an aggregate function)  UPDATE DepartmentSalaryStats  SET avg\_salary = avg\_salary + 5000  WHERE department\_name = 'Sales'; -- This will cause an error  -- 8. Create a view with an insertable join (this will allow updates if certain conditions are met)  CREATE VIEW Employees\_Departments\_Insert\_View AS  SELECT e.employee\_id, e.employee\_name, e.salary, d.department\_name  FROM Employees e  JOIN Departments d ON e.department\_id = d.department\_id;  -- Query the view before insertion  SELECT \* FROM Employees\_Departments\_Insert\_View;  -- 9. Insert into Employees via the insertable view  INSERT INTO Employees\_Departments\_Insert\_View (employee\_id, employee\_name, salary, department\_name)  VALUES (6, 'Samuel Green', 75000, 'Engineering');  -- Verify the insertion in the base table (Employees)  SELECT \* FROM Employees WHERE employee\_name = 'Samuel Green';  -- 10. Check the effect on the view after the insertion  SELECT \* FROM Employees\_Departments\_Insert\_View;  -- Cleanup: Drop the views and tables after testing  DROP VIEW IF EXISTS Employees\_View;  DROP VIEW IF EXISTS Employees\_Departments\_View;  DROP VIEW IF EXISTS HighSalaryEmployees\_View;  DROP VIEW IF EXISTS DepartmentSalaryStats;  DROP VIEW IF EXISTS Employees\_Departments\_Insert\_View;  DROP TABLE IF EXISTS Employees;  DROP TABLE IF EXISTS Departments; |
| 19 | Discuss the strengths and weaknesses of the trigger mechanism. Contrast triggers with other integrity constraints supported by SQL. |
|  | -- Drop existing tables if they exist  DROP TABLE IF EXISTS Employees;  DROP TABLE IF EXISTS Departments;  DROP TABLE IF EXISTS Salary\_Audit;  -- Create Employees table  CREATE TABLE Employees (  employee\_id INT PRIMARY KEY,  employee\_name VARCHAR(100),  department\_id INT,  salary REAL,  hire\_date DATE  );  -- Create Departments table  CREATE TABLE Departments (  department\_id INT PRIMARY KEY,  department\_name VARCHAR(100),  manager\_id INT  );  -- Create Salary Audit table for auditing salary changes  CREATE TABLE Salary\_Audit (  audit\_id INT AUTO\_INCREMENT PRIMARY KEY,  employee\_id INT,  old\_salary REAL,  new\_salary REAL,  change\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );  -- Insert sample data into Employees table  INSERT INTO Employees (employee\_id, employee\_name, department\_id, salary, hire\_date) VALUES  (1, 'John Doe', 101, 50000, '2020-05-15'),  (2, 'Jane Smith', 102, 60000, '2019-07-30'),  (3, 'Mark Johnson', 101, 55000, '2018-03-20'),  (4, 'Lucy Brown', 103, 70000, '2021-08-10');  -- Insert sample data into Departments table  INSERT INTO Departments (department\_id, department\_name, manager\_id) VALUES  (101, 'Sales', 1),  (102, 'Marketing', 2),  (103, 'Engineering', 4);  -- 1. Trigger for automatically updating the salary and recording audit when salary is updated  DELIMITER $$  CREATE TRIGGER before\_salary\_update  BEFORE UPDATE ON Employees  FOR EACH ROW  BEGIN  -- Insert old salary and new salary into the Salary\_Audit table  INSERT INTO Salary\_Audit (employee\_id, old\_salary, new\_salary)  VALUES (OLD.employee\_id, OLD.salary, NEW.salary);    -- Automatically give a raise to the employee when their salary is updated  IF NEW.salary < 100000 THEN  SET NEW.salary = NEW.salary + 5000;  END IF;  END $$  DELIMITER ;  -- Test: Update salary for an employee  UPDATE Employees  SET salary = 95000  WHERE employee\_name = 'John Doe';  -- Check Salary\_Audit table for audit log  SELECT \* FROM Salary\_Audit;  -- Check the updated salary in Employees table  SELECT \* FROM Employees WHERE employee\_name = 'John Doe';  -- 2. Trigger for ensuring managers have higher salaries than their employees  DELIMITER $$  CREATE TRIGGER check\_manager\_salary  BEFORE INSERT ON Employees  FOR EACH ROW  BEGIN  DECLARE manager\_salary REAL;    -- Get the salary of the manager of the employee's department  SELECT salary INTO manager\_salary  FROM Employees  WHERE employee\_id = (SELECT manager\_id FROM Departments WHERE department\_id = NEW.department\_id);    -- Ensure that the employee's salary is less than the manager's salary  IF NEW.salary >= manager\_salary THEN  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT = 'Employee salary cannot be greater than or equal to manager salary';  END IF;  END $$  DELIMITER ;  -- Test: Insert a new employee with a salary greater than their manager  INSERT INTO Employees (employee\_id, employee\_name, department\_id, salary, hire\_date)  VALUES (5, 'Samuel Green', 101, 120000, '2022-01-25');  -- The above insert will fail due to the trigger  -- Cleanup: Drop the triggers and tables after testing  DROP TRIGGER IF EXISTS before\_salary\_update;  DROP TRIGGER IF EXISTS check\_manager\_salary;  DROP TABLE IF EXISTS Employees;  DROP TABLE IF EXISTS Departments;  DROP TABLE IF EXISTS Salary\_Audit; |
| 20 | 1. Write a PL/SQL block that implements Function.  2. Write a PL/SQL block that implements AFTER INSERT TRIGGER.  3. Write a PL/SQL block that implements BEFORE DELETE TRIGGER  4. Write a trigger to check the salary is not Zero or Negative.  5. Write a Trigger that check the employee resides in a city 'Pune' |
|  | -- 1. PL/SQL Block that Implements a Function  -- Function to calculate bonus based on employee salary  CREATE OR REPLACE FUNCTION calculate\_bonus (emp\_salary IN NUMBER)  RETURN NUMBER IS  bonus NUMBER;  BEGIN  -- Calculate bonus: 10% of salary  bonus := emp\_salary \* 0.1;  RETURN bonus;  END;  /  -- Example usage of the function:  DECLARE  emp\_bonus NUMBER;  BEGIN  -- Assume an employee salary of 50000  emp\_bonus := calculate\_bonus(50000);  DBMS\_OUTPUT.PUT\_LINE('Employee Bonus: ' || emp\_bonus);  END;  /  -- 2. PL/SQL Block that Implements AFTER INSERT TRIGGER  -- Trigger that fires after inserting a new employee to log information in audit table  CREATE OR REPLACE TRIGGER after\_employee\_insert  AFTER INSERT ON Employees  FOR EACH ROW  BEGIN  -- Assuming there is an audit table to log changes  INSERT INTO Employee\_Audit (emp\_id, action, timestamp)  VALUES (:NEW.employee\_id, 'INSERT', SYSDATE);  DBMS\_OUTPUT.PUT\_LINE('Audit log created for INSERT operation');  END;  /  -- Example Insert to test the AFTER INSERT trigger:  -- Inserting a new employee will trigger the audit log insertion  INSERT INTO Employees (employee\_id, employee\_name, salary, department\_id)  VALUES (1, 'John Doe', 50000, 101);  -- 3. PL/SQL Block that Implements BEFORE DELETE TRIGGER  -- Trigger to check if an employee exists before deleting  CREATE OR REPLACE TRIGGER before\_employee\_delete  BEFORE DELETE ON Employees  FOR EACH ROW  BEGIN  -- Check if the employee exists in the Employees table before deletion  IF :OLD.employee\_id IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Employee does not exist');  END IF;  DBMS\_OUTPUT.PUT\_LINE('Employee deletion is about to occur');  END;  /  -- Example Delete to test the BEFORE DELETE trigger:  -- Deleting an employee will trigger the check  DELETE FROM Employees WHERE employee\_id = 1;  -- 4. Trigger to Check the Salary is Not Zero or Negative  CREATE OR REPLACE TRIGGER check\_salary  BEFORE INSERT OR UPDATE ON Employees  FOR EACH ROW  BEGIN  IF :NEW.salary <= 0 THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Salary cannot be zero or negative');  END IF;  DBMS\_OUTPUT.PUT\_LINE('Salary is valid');  END;  /  -- Example Insert to test the salary check:  -- This insert will fail because salary is zero  INSERT INTO Employees (employee\_id, employee\_name, salary, department\_id)  VALUES (2, 'Jane Smith', 0, 102);  -- 5. Trigger to Check if the Employee Resides in 'Pune'  CREATE OR REPLACE TRIGGER check\_employee\_city  BEFORE INSERT OR UPDATE ON Employees  FOR EACH ROW  BEGIN  IF :NEW.city != 'Pune' THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Employee must reside in Pune');  END IF;  DBMS\_OUTPUT.PUT\_LINE('Employee resides in Pune');  END;  /  -- Example Insert to test the city check:  -- This insert will fail because city is not 'Pune'  INSERT INTO Employees (employee\_id, employee\_name, salary, department\_id, city)  VALUES (3, 'Mark Johnson', 60000, 103, 'Mumbai'); |
| 21 | Manipulating Data  (1) Give 20% interest to all depositors.  (2) Give 10% interest to all depositors having branch Manjari  (3) Give 10% interest to all depositors living in Pune and having branch city bombay.  (4) Write a query which changes the department number of all employees with empno7788’s job to employee 7844’current department number  (5) Write a query which changes the department number of all employees with empno.  (6) Transfer 10 Rs from account of anil to sunil if both are having same branch  (7) Give 100 Rs more to all depositors if they are maximum depositors in their respective branch.  (8) Delete deposit of vijay  (9) Delete borrower of branches having average loan less than 1000 |
|  | -- Step 1: Create the Database  CREATE DATABASE IF NOT EXISTS BankDB;  USE BankDB;  -- Step 2: Create Tables  -- Table to store depositor information  CREATE TABLE Depositors (  id INT AUTO\_INCREMENT PRIMARY KEY,  name VARCHAR(50),  balance DECIMAL(15, 2),  branch VARCHAR(50),  city VARCHAR(50)  );  -- Table to store employee information  CREATE TABLE Employees (  empno INT PRIMARY KEY,  ename VARCHAR(50),  job VARCHAR(50),  salary DECIMAL(15, 2),  deptno INT  );  -- Table to store loan details  CREATE TABLE Loans (  loan\_id INT AUTO\_INCREMENT PRIMARY KEY,  branch VARCHAR(50),  loan\_amount DECIMAL(15, 2)  );  -- Table to store borrower information  CREATE TABLE Borrowers (  borrower\_id INT AUTO\_INCREMENT PRIMARY KEY,  name VARCHAR(50),  branch VARCHAR(50)  );  -- Step 3: Insert Sample Data  -- Inserting sample data into Depositors  INSERT INTO Depositors (name, balance, branch, city)  VALUES  ('Anil', 5000.00, 'Mumbai', 'Pune'),  ('Sunil', 3000.00, 'Mumbai', 'Pune'),  ('Vijay', 15000.00, 'Manjari', 'Pune'),  ('Raj', 12000.00, 'Bombay', 'Mumbai'),  ('Meera', 8000.00, 'Bombay', 'Delhi');  -- Inserting sample data into Employees  INSERT INTO Employees (empno, ename, job, salary, deptno)  VALUES  (7788, 'John', 'Manager', 80000.00, 10),  (7844, 'Mark', 'Engineer', 60000.00, 20),  (7934, 'Sophia', 'Clerk', 40000.00, 30),  (7566, 'Alice', 'Manager', 85000.00, 10),  (7698, 'Bob', 'Analyst', 50000.00, 20);  -- Inserting sample data into Loans  INSERT INTO Loans (branch, loan\_amount)  VALUES  ('Mumbai', 2000.00),  ('Pune', 500.00),  ('Bombay', 1500.00),  ('Manjari', 3000.00);  -- Inserting sample data into Borrowers  INSERT INTO Borrowers (name, branch)  VALUES  ('Anil', 'Mumbai'),  ('Vijay', 'Pune'),  ('Raj', 'Bombay'),  ('Meera', 'Manjari');  -- Step 4: Data Manipulation Queries  -- 1. Give 20% interest to all depositors.  UPDATE Depositors  SET balance = balance \* 1.20;  -- 2. Give 10% interest to all depositors having branch 'Manjari'.  UPDATE Depositors  SET balance = balance \* 1.10  WHERE branch = 'Manjari';  -- 3. Give 10% interest to all depositors living in Pune and having branch 'Bombay'.  UPDATE Depositors  SET balance = balance \* 1.10  WHERE city = 'Pune' AND branch = 'Bombay';  -- 4. Write a query which changes the department number of all employees with empno 7788’s job to employee 7844’s current department number.  UPDATE Employees  SET deptno = (SELECT deptno FROM Employees WHERE empno = 7844)  WHERE job = (SELECT job FROM Employees WHERE empno = 7788);  -- 5. Write a query which changes the department number of all employees with empno 7788.  UPDATE Employees  SET deptno = (SELECT deptno FROM Employees WHERE empno = 7788)  WHERE empno = 7788;  -- 6. Transfer 10 Rs from Anil's account to Sunil's if both are in the same branch.  UPDATE Depositors d1, Depositors d2  SET d1.balance = d1.balance - 10, d2.balance = d2.balance + 10  WHERE d1.name = 'Anil' AND d2.name = 'Sunil' AND d1.branch = d2.branch;  -- 7. Give 100 Rs to maximum depositors in their respective branches.  UPDATE Depositors d1  SET d1.balance = d1.balance + 100  WHERE d1.balance = (SELECT MAX(balance) FROM Depositors d2 WHERE d2.branch = d1.branch);  -- 8. Delete the deposit of 'Vijay'.  DELETE FROM Depositors  WHERE name = 'Vijay';  -- 9. Delete borrowers of branches having average loan less than 1000.  DELETE FROM Borrowers  WHERE branch IN (  SELECT branch FROM Loans  GROUP BY branch  HAVING AVG(loan\_amount) < 1000  ); |
| 22 | Create a database with suitable example using MongoDB and implement   Inserting and saving document (batch insert, insert validation)   Removing document   Updating document (document replacement, using modifiers, upserts, updating  documents, returning updated documents) |
|  | // Step 1: Create or switch to the database  use BankDB; // Use or create a database called BankDB  // Step 2: Create a collection and insert documents  // Create a collection called 'customers' and insert some documents  db.customers.insertMany([  { name: "John Doe", balance: 5000, city: "New York", active: true },  { name: "Jane Smith", balance: 3000, city: "Los Angeles", active: true },  { name: "Robert Brown", balance: 7000, city: "Chicago", active: false },  { name: "Emily Davis", balance: 6000, city: "Houston", active: true }  ]);  // Step 3: Insert a single document  db.customers.insertOne({  name: "Alice Walker",  balance: 4000,  city: "Boston",  active: true  });  // Step 4: Remove a document based on a condition  db.customers.deleteOne({ name: "Robert Brown" });  // Step 5: Update a document using $set to change balance  db.customers.updateOne(  { name: "Emily Davis" }, // Condition  { $set: { balance: 6500 } } // Update balance to 6500  );  // Step 6: Update multiple documents (increase balance by 1000 for active users)  db.customers.updateMany(  { active: true }, // Condition  { $inc: { balance: 1000 } } // Increment balance by 1000  );  // Step 7: Find and display all documents in the collection  print("All Customers:");  db.customers.find().pretty(); // Pretty format to display all customers  // Step 8: Update a document and return the updated one  var updatedDoc = db.customers.findOneAndUpdate(  { name: "Jane Smith" }, // Condition  { $set: { balance: 3500 } }, // Update balance  { returnNewDocument: true } // Return the updated document  );  printjson(updatedDoc); // Print the updated document  // Step 9: Insert a new document that will violate validation rules (if any)  try {  db.customers.insertOne({  name: "Invalid Entry", // Missing balance (violates validation rules)  city: "Delhi",  active: true  });  } catch (e) {  print("Error: " + e); // Print error if validation fails  } |
| 23 | Execute at least 10 queries on any suitable MongoDB database that demonstrates following querying  techniques:   find and findOne (specific values)   $gt, $lt   delete and deleteOne |
|  | // 1. Create and Switch to the Database  use companyDB;  // 2. Insert Sample Data into the 'employees' Collection  db.employees.insertMany([  { \_id: 1, name: "John Doe", age: 29, salary: 55000, department: "HR", city: "New York" },  { \_id: 2, name: "Jane Smith", age: 34, salary: 72000, department: "Engineering", city: "Chicago" },  { \_id: 3, name: "Alice Brown", age: 25, salary: 48000, department: "Marketing", city: "New York" },  { \_id: 4, name: "Bob White", age: 42, salary: 95000, department: "Engineering", city: "San Francisco" },  { \_id: 5, name: "Charlie Black", age: 37, salary: 82000, department: "Sales", city: "Chicago" },  { \_id: 6, name: "David Green", age: 31, salary: 67000, department: "Marketing", city: "New York" }  ]);  // 3. Verify the Data Insertion  print("All employees:");  db.employees.find().pretty();  // 4. Query 1: Using findOne to get a specific employee by name  print("Find employee with name 'Jane Smith':");  var employee = db.employees.findOne({ name: "Jane Smith" });  printjson(employee);  // 5. Query 2: Using find to get employees with salary greater than 60,000  print("Employees with salary greater than 60000:");  db.employees.find({  salary: { $gt: 60000 }  }).pretty();  // 6. Query 3: Using find to get employees with salary less than 60,000  print("Employees with salary less than 60000:");  db.employees.find({  salary: { $lt: 60000 }  }).pretty();  // 7. Query 4: Using find to get employees in "Engineering" department  print("Employees in Engineering department:");  db.employees.find({  department: "Engineering"  }).pretty();  // 8. Query 5: Update an Employee's Salary (using $set modifier)  print("Updating Bob White's salary to 105000:");  db.employees.updateOne(  { name: "Bob White" },  { $set: { salary: 105000 } }  );  // 9. Query 6: Find all employees whose salary is between 50,000 and 80,000 (using $gt and $lt)  print("Employees with salary between 50000 and 80000:");  db.employees.find({  salary: { $gt: 50000, $lt: 80000 }  }).pretty();  // 10. Query 7: Using deleteOne to delete a specific employee  print("Deleting employee with name 'Alice Brown':");  db.employees.deleteOne({ name: "Alice Brown" });  // 11. Query 8: Using delete to delete all employees from New York  print("Deleting all employees from New York:");  db.employees.deleteMany({ city: "New York" });  // 12. Query 9: Update all employees from "Chicago" to increase salary by 10%  print("Increasing salary of employees from Chicago by 10%:");  db.employees.updateMany(  { city: "Chicago" },  { $mul: { salary: 1.10 } }  );  // 13. Query 10: Find employees older than 30 using $gt (greater than)  print("Employees older than 30:");  db.employees.find({  age: { $gt: 30 }  }).pretty();  // 14. Verify Changes  print("All employees after operations:");  db.employees.find().pretty(); |
| 24 | Execute at least 10 queries on any suitable MongoDB database that demonstrates following:   $ where queries   CRUD Database commands |
|  | // 1. Create and Switch to the Database  use companyDB;  // 2. Insert Sample Data into the 'employees' Collection  db.employees.insertMany([  { \_id: 1, name: "John Doe", age: 29, salary: 55000, department: "HR", city: "New York" },  { \_id: 2, name: "Jane Smith", age: 34, salary: 72000, department: "Engineering", city: "Chicago" },  { \_id: 3, name: "Alice Brown", age: 25, salary: 48000, department: "Marketing", city: "New York" },  { \_id: 4, name: "Bob White", age: 42, salary: 95000, department: "Engineering", city: "San Francisco" },  { \_id: 5, name: "Charlie Black", age: 37, salary: 82000, department: "Sales", city: "Chicago" },  { \_id: 6, name: "David Green", age: 31, salary: 67000, department: "Marketing", city: "New York" }  ]);  // 3. Verify the Data Insertion  print("All employees:");  db.employees.find().pretty();  // 4. Query 1: $where Query - Salary greater than 60,000  print("Employees with salary greater than 60000:");  db.employees.find({  $where: function() {  return this.salary > 60000;  }  }).pretty();  // 5. Query 2: $where Query - Age greater than 30 and city is "New York"  print("Employees older than 30 and from New York:");  db.employees.find({  $where: function() {  return this.age > 30 && this.city === "New York";  }  }).pretty();  // 6. Query 3: Find Employees with Salary between 50,000 and 80,000 (using $lt and $gt)  print("Employees with salary between 50000 and 80000:");  db.employees.find({  salary: { $gt: 50000, $lt: 80000 }  }).pretty();  // 7. Query 4: Update an Employee's Salary using the $set modifier  print("Updating Bob White's salary to 105000:");  db.employees.updateOne(  { name: "Bob White" },  { $set: { salary: 105000 } }  );  // 8. Query 5: Delete an Employee based on \_id  print("Deleting Alice Brown from the employees collection:");  db.employees.deleteOne({ \_id: 3 });  // 9. Query 6: Find Employees from "Chicago" and update their salary by 10%  print("Increasing salary of employees from Chicago by 10%:");  db.employees.updateMany(  { city: "Chicago" },  { $mul: { salary: 1.10 } }  );  // 10. Query 7: Add a new employee using insertOne  print("Adding new employee:");  db.employees.insertOne({  \_id: 7,  name: "Eva White",  age: 29,  salary: 60000,  department: "HR",  city: "Chicago"  });  // 11. Query 8: Upsert an Employee (insert if not exists, update if exists)  print("Upserting employee with id 7:");  db.employees.updateOne(  { \_id: 7 },  { $set: { salary: 65000, department: "HR" } },  { upsert: true }  );  // 12. Query 9: Find Employees with age greater than 30 and city as "New York"  print("Employees older than 30 and from New York:");  db.employees.find({  $where: function() {  return this.age > 30 && this.city === "New York";  }  }).pretty();  // 13. Query 10: Delete all employees in "San Francisco" who are older than 40  print("Deleting employees from San Francisco who are older than 40:");  db.employees.deleteMany({  city: "San Francisco",  age: { $gt: 40 }  }); |
| 25 | Mongodb CRUD operations |
|  | // Step 1: Create and Switch to the Database  use schoolDB; // Create and use a database called "schoolDB"  // Step 2: Create a Collection and Insert Data (Create Operation)  db.students.insertMany([  { \_id: 1, name: "John Doe", age: 18, grade: "A", city: "New York" },  { \_id: 2, name: "Jane Smith", age: 19, grade: "B", city: "Los Angeles" },  { \_id: 3, name: "Alice Brown", age: 20, grade: "A", city: "Chicago" },  { \_id: 4, name: "Bob White", age: 17, grade: "C", city: "Houston" },  { \_id: 5, name: "Charlie Black", age: 21, grade: "B", city: "Phoenix" }  ]);  // Verify the data inserted  print("All Students:");  db.students.find().pretty();  // Step 3: Read Operations (Find Queries)  // 1. Find all students in the collection  print("Find All Students:");  db.students.find().pretty();  // 2. Find a specific student by name (findOne)  print("Find Student with Name 'John Doe':");  var student = db.students.findOne({ name: "John Doe" });  printjson(student);  // 3. Find students with grade "A"  print("Find Students with Grade 'A':");  db.students.find({ grade: "A" }).pretty();  // Step 4: Update Operations  // 1. Update the grade of a student (updateOne)  print("Update Bob White's Grade to 'B':");  db.students.updateOne(  { name: "Bob White" },  { $set: { grade: "B" } }  );  // 2. Update the age of students who live in "New York" (updateMany)  print("Increase Age of Students from New York by 1:");  db.students.updateMany(  { city: "New York" },  { $inc: { age: 1 } }  );  // Step 5: Delete Operations  // 1. Delete a specific student by name (deleteOne)  print("Delete Student 'Charlie Black':");  db.students.deleteOne({ name: "Charlie Black" });  // 2. Delete all students from the city "Los Angeles" (deleteMany)  print("Delete all Students from Los Angeles:");  db.students.deleteMany({ city: "Los Angeles" });  // Verify data after operations  print("All Students after CRUD Operations:");  db.students.find().pretty(); |
| 26 | Implement the aggregation and indexing with suitable example in MongoDB. Demonstrate the  following:   Aggregation framework   Create and drop different types of indexes and explain () to show the advantage of the indexes. |
|  | // Step 1: Create and Use a Database  use schoolDB; // Create and use a database called "schoolDB"  // Step 2: Create a Collection and Insert Data (Create Operation)  db.students.insertMany([  { \_id: 1, name: "John Doe", age: 18, grade: "A", city: "New York", score: 90 },  { \_id: 2, name: "Jane Smith", age: 19, grade: "B", city: "Los Angeles", score: 75 },  { \_id: 3, name: "Alice Brown", age: 20, grade: "A", city: "Chicago", score: 85 },  { \_id: 4, name: "Bob White", age: 17, grade: "C", city: "Houston", score: 60 },  { \_id: 5, name: "Charlie Black", age: 21, grade: "B", city: "Phoenix", score: 80 }  ]);  // Verify the data inserted  print("All Students:");  db.students.find().pretty();  // Step 3: Aggregation Framework Example  // 1. Grouping by Grade and calculating average score  print("Aggregation: Group by Grade and Calculate Average Score:");  db.students.aggregate([  {  $group: {  \_id: "$grade", // Group by the "grade" field  averageScore: { $avg: "$score" } // Calculate the average score for each grade  }  }  ]);  // 2. Filter students who have a score greater than 80 and group by City  print("Aggregation: Filter students with score > 80 and group by City:");  db.students.aggregate([  { $match: { score: { $gt: 80 } } }, // Match students with score > 80  {  $group: {  \_id: "$city", // Group by city  totalStudents: { $sum: 1 }, // Count the total number of students in each city  avgScore: { $avg: "$score" } // Calculate the average score in each city  }  }  ]);  // 3. Sorting the results by Score in descending order  print("Aggregation: Sort by Score (Descending):");  db.students.aggregate([  { $sort: { score: -1 } } // Sort by score in descending order  ]);  // 4. Projecting only the name and score of students  print("Aggregation: Project Name and Score Only:");  db.students.aggregate([  { $project: { name: 1, score: 1 } } // Only display name and score  ]);  // Step 4: Indexing Examples  // 1. Create a Single Field Index on the "city" field  print("Create Single Field Index on 'city':");  db.students.createIndex({ city: 1 });  // 2. Create a Compound Index on "grade" and "score" fields  print("Create Compound Index on 'grade' and 'score':");  db.students.createIndex({ grade: 1, score: -1 });  // 3. Create a Text Index on the "name" field  print("Create Text Index on 'name' for text search:");  db.students.createIndex({ name: "text" });  // 4. Create a Geospatial Index on a "location" field (assumed to be in your schema)  print("Create Geospatial Index on 'location' field (if the field exists):");  db.students.createIndex({ location: "2dsphere" });  // Step 5: Demonstrating the Advantage of Indexes  // Query without using index (slow)  print("Query without index (Full Collection Scan):");  var start = new Date();  db.students.find({ city: "New York" }).toArray();  var end = new Date();  print("Query time without index (ms): " + (end - start));  // Query using index (faster)  print("Query with index (Using the 'city' index):");  start = new Date();  db.students.find({ city: "New York" }).hint({ city: 1 }).toArray();  end = new Date();  print("Query time with index (ms): " + (end - start));  // Step 6: Drop Indexes  // Drop the created index on the 'city' field  print("Drop the Index on 'city':");  db.students.dropIndex({ city: 1 });  // Step 7: Verify Data After Operations  print("All Students after Index Operations:");  db.students.find().pretty(); |
| 27 | Implement nested sub queries. Perform a test for set membership (in, not in), set comparison  (<some, >=some, <all etc.) and set cardinality (unique, not unique).  Assume suitable tables. |
|  | // Step 1: Create a new database  use storeDB;  // Step 2: Create collections and insert sample data  db.products.insertMany([  { \_id: 1, name: "Laptop", price: 800, category: "Electronics" },  { \_id: 2, name: "Tablet", price: 300, category: "Electronics" },  { \_id: 3, name: "Smartphone", price: 600, category: "Electronics" },  { \_id: 4, name: "Shirt", price: 25, category: "Clothing" },  { \_id: 5, name: "Jeans", price: 40, category: "Clothing" }  ]);  db.orders.insertMany([  { order\_id: 101, customer\_name: "Alice", product\_id: 1, quantity: 1 },  { order\_id: 102, customer\_name: "Bob", product\_id: 3, quantity: 2 },  { order\_id: 103, customer\_name: "Charlie", product\_id: 2, quantity: 1 },  { order\_id: 104, customer\_name: "David", product\_id: 5, quantity: 3 }  ]);  // Step 3: Execute nested subqueries and set operations  // Example 1: Using `$in` (Set Membership) - Find orders for products with ids 1 and 2  db.orders.find({ product\_id: { $in: [1, 2] } });  // Example 2: Using `$gt` and `$lt` (Set Comparison) - Find products with price greater than 500  db.products.find({ price: { $gt: 500 } });  // Example 3: Using `$ne` (Set Comparison) - Find products that are NOT Electronics  db.products.find({ category: { $ne: "Electronics" } });  // Example 4: Using `$nin` (Set Membership) - Find orders for products not in the list [1, 3]  db.orders.find({ product\_id: { $nin: [1, 3] } });  // Example 5: Using `$all` (Set Cardinality) - Find products in both Electronics and Clothing categories  db.products.find({ category: { $all: ["Electronics", "Clothing"] } });  // Example 6: Using `$size` (Set Cardinality) - Find orders with exactly 2 products  db.orders.aggregate([  { $lookup: {  from: "products",  localField: "product\_id",  foreignField: "\_id",  as: "product\_details"  }},  { $match: { "product\_details": { $size: 2 } } }  ]);  // Example 7: Nested Query - Find customers who ordered products with price greater than 500  db.orders.find({  product\_id: {  $in: db.products.find({ price: { $gt: 500 } }).map(product => product.\_id)  }  }); |
| 28 | Execute DDL statements which demonstrate the use of views. Try to update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables. |
|  | -- Step 1: Create a database  CREATE DATABASE IF NOT EXISTS StoreDB;  -- Use the created database  USE StoreDB;  -- Step 2: Create base tables (Products and Orders)  CREATE TABLE IF NOT EXISTS Products (  product\_id INT PRIMARY KEY AUTO\_INCREMENT,  product\_name VARCHAR(255) NOT NULL,  price DECIMAL(10, 2) NOT NULL,  category VARCHAR(255)  );  CREATE TABLE IF NOT EXISTS Orders (  order\_id INT PRIMARY KEY AUTO\_INCREMENT,  product\_id INT,  customer\_name VARCHAR(255),  order\_date DATE,  quantity INT,  FOREIGN KEY (product\_id) REFERENCES Products(product\_id)  );  -- Step 3: Insert some data into Products and Orders tables  INSERT INTO Products (product\_name, price, category) VALUES  ('Laptop', 800.00, 'Electronics'),  ('Smartphone', 500.00, 'Electronics'),  ('Shirt', 30.00, 'Clothing');  INSERT INTO Orders (product\_id, customer\_name, order\_date, quantity) VALUES  (1, 'Alice', '2023-10-15', 2),  (2, 'Bob', '2023-10-16', 1),  (3, 'Charlie', '2023-10-17', 3);  -- Step 4: Create a simple view from a single table  CREATE VIEW ProductInfo AS  SELECT product\_id, product\_name, price, category  FROM Products;  -- Query the view  SELECT \* FROM ProductInfo;  -- Step 5: Create a view that joins multiple tables  CREATE VIEW OrderDetails AS  SELECT o.order\_id, o.product\_id, o.customer\_name, o.order\_date, o.quantity, p.product\_name, p.price  FROM Orders o  JOIN Products p ON o.product\_id = p.product\_id;  -- Query the view  SELECT \* FROM OrderDetails;  -- Step 6: Update the base table through a view (View based on a single table)  UPDATE ProductInfo  SET price = 850.00  WHERE product\_name = 'Laptop';  -- Check the updated Products table  SELECT \* FROM Products;  -- Step 7: Try updating the base table through a view with JOIN (This will not work as expected)  -- This will fail because the view involves multiple tables and is not updatable.  UPDATE OrderDetails  SET quantity = 5  WHERE order\_id = 1;  -- Error will occur: You can't update a view based on multiple tables directly  -- Step 8: Create an updatable view by simplifying the view to a single table  CREATE VIEW UpdatableProductView AS  SELECT product\_id, product\_name, price  FROM Products;  -- Update the base table via the updatable view  UPDATE UpdatableProductView  SET price = 900.00  WHERE product\_name = 'Smartphone';  -- Check the updated Products table  SELECT \* FROM Products;  -- Step 9: Drop the views when done  DROP VIEW IF EXISTS ProductInfo;  DROP VIEW IF EXISTS OrderDetails;  DROP VIEW IF EXISTS UpdatableProductView;  -- Step 10: Clean up (optional, for good practice)  DROP DATABASE IF EXISTS StoreDB; |
| 29 | Create a database with suitable example using MongoDB and implement   Inserting and saving document (batch insert, insert validation)   Removing document   Updating document |
|  |  |
|  | // Connect to MongoDB and create a database  use MyDatabase; // Switch to the database named 'MyDatabase' (it will be created if it doesn't exist)  // Step 1: Create collections  db.createCollection("customers");  db.createCollection("orders");  // Step 2: Inserting documents  // Insert a single document into 'customers' collection  db.customers.insertOne({  customer\_id: 1,  name: "John Doe",  email: "johndoe@example.com",  age: 30  });  // Batch Insert documents into 'customers' collection  db.customers.insertMany([  {  customer\_id: 2,  name: "Jane Smith",  email: "janesmith@example.com",  age: 25  },  {  customer\_id: 3,  name: "Michael Johnson",  email: "michaelj@example.com",  age: 35  }  ]);  // Step 3: Insert with validation (e.g., ensuring email uniqueness)  db.createCollection("orders", {  validator: {  $jsonSchema: {  bsonType: "object",  required: ["order\_id", "customer\_id", "product", "quantity"],  properties: {  order\_id: {  bsonType: "int",  description: "must be an integer and is required"  },  customer\_id: {  bsonType: "int",  description: "must be an integer and is required"  },  product: {  bsonType: "string",  description: "must be a string and is required"  },  quantity: {  bsonType: "int",  description: "must be an integer and is required"  }  }  }  }  });  // Insert a document into 'orders' collection (with validation)  db.orders.insertOne({  order\_id: 1,  customer\_id: 1,  product: "Laptop",  quantity: 2  });  // Step 4: Remove document  // Remove a single document based on condition  db.customers.deleteOne({ customer\_id: 2 });  // Remove multiple documents based on condition  db.customers.deleteMany({ age: { $gt: 30 } }); // Deletes customers older than 30  // Step 5: Updating documents  // Update a single document  db.customers.updateOne(  { customer\_id: 1 }, // Find the customer with ID 1  { $set: { email: "john.doe@newdomain.com" } } // Update email  );  // Update multiple documents  db.customers.updateMany(  { age: { $gt: 30 } }, // Find customers older than 30  { $set: { status: "Premium" } } // Add or update 'status' field to "Premium"  );  // Step 6: Using the find() method to check the current state  db.customers.find().pretty(); // Display all customers  db.orders.find().pretty(); // Display all orders  // Step 7: Updating using $inc (increase a field value)  db.customers.updateOne(  { customer\_id: 1 }, // Find the customer with ID 1  { $inc: { age: 1 } } // Increase age by 1  );  // Step 8: Updating embedded fields  // Suppose we add an address field inside the customer document:  db.customers.updateOne(  { customer\_id: 1 },  { $set: { "address.city": "New York", "address.zip": "10001" } } // Add nested fields for address  );  // Check the updated document  db.customers.find({ customer\_id: 1 }).pretty(); |

|  |  |
| --- | --- |
| 30 | Consider the table and solve the quries:  Sailors( sid, sname, rating, age)  Boats(bid, bname,color)  Reserves(sid,bid,day)   1. Find the names of sailors who have reserved boat number 103 2. Find the names of sailors who have never reserved boat number 103 3. Find the names of sailors who have reserved a red boat 4. Find the colors of boats reserved by Lubber 5. Find the names of sailors who have reserved at least one boat |
|  |  |
|  | -- Step 1: Create the database  CREATE DATABASE SailorDB;  -- Step 2: Use the database  USE SailorDB;  -- Step 3: Create tables  CREATE TABLE Sailors (  sid INT PRIMARY KEY,  sname VARCHAR(50),  rating INT,  age INT  );  CREATE TABLE Boats (  bid INT PRIMARY KEY,  bname VARCHAR(50),  color VARCHAR(20)  );  CREATE TABLE Reserves (  sid INT,  bid INT,  day DATE,  PRIMARY KEY (sid, bid, day),  FOREIGN KEY (sid) REFERENCES Sailors(sid),  FOREIGN KEY (bid) REFERENCES Boats(bid)  );  -- Step 4: Insert sample data into the tables  -- Insert data into Sailors  INSERT INTO Sailors (sid, sname, rating, age) VALUES  (1, 'Lubber', 5, 30),  (2, 'Penny', 7, 25),  (3, 'Sam', 4, 35),  (4, 'Tom', 6, 28);  -- Insert data into Boats  INSERT INTO Boats (bid, bname, color) VALUES  (101, 'Red Boat', 'red'),  (102, 'Blue Boat', 'blue'),  (103, 'Green Boat', 'green'),  (104, 'Yellow Boat', 'yellow');  -- Insert data into Reserves  INSERT INTO Reserves (sid, bid, day) VALUES  (1, 103, '2024-11-01'),  (1, 102, '2024-11-02'),  (2, 101, '2024-11-03'),  (3, 103, '2024-11-04'),  (4, 101, '2024-11-05');  -- Step 5: Run the queries  -- Query 1: Find the names of sailors who have reserved boat number 103.  SELECT s.sname  FROM Sailors s  JOIN Reserves r ON s.sid = r.sid  WHERE r.bid = 103;  -- Query 2: Find the names of sailors who have never reserved boat number 103.  SELECT s.sname  FROM Sailors s  WHERE s.sid NOT IN (  SELECT r.sid  FROM Reserves r  WHERE r.bid = 103  );  -- Query 3: Find the names of sailors who have reserved a red boat.  SELECT DISTINCT s.sname  FROM Sailors s  JOIN Reserves r ON s.sid = r.sid  JOIN Boats b ON r.bid = b.bid  WHERE b.color = 'red';  -- Query 4: Find the colors of boats reserved by Lubber.  SELECT DISTINCT b.color  FROM Boats b  JOIN Reserves r ON b.bid = r.bid  JOIN Sailors s ON r.sid = s.sid  WHERE s.sname = 'Lubber';  -- Query 5: Find the names of sailors who have reserved at least one boat.  SELECT DISTINCT s.sname  FROM Sailors s  JOIN Reserves r ON s.sid = r.sid; |
|  |  |
| 31 | Consider the table and solve the quries:  Sailors( sid, sname, rating, age)  Boats(bid, bname,color)  Reserves(sid,bid,day)   1. Find the names of sailors who have reserved a red or a green boat 2. Find the names of sailors who have reserved both a red and a green boat 3. Find the names of sailors who have reserved at least two different boats 4. Find the sids of silors with age over 20 who have not reserved a red boat 5. Find the names of sailors who have reserved all boats 6. Find the names of sailors who have reserved all boats called Interlake 7. Find all sailors with a rating above 7 8. Find the sailor name boat id and reservation date for each reservation 9. Find sailors who have reserved all red boats 10. Find the names and ages of all sailors |
|  | -- Step 1: Create the database  CREATE DATABASE SailorDB;  -- Step 2: Use the database  USE SailorDB;  -- Step 3: Create tables  CREATE TABLE Sailors (  sid INT PRIMARY KEY,  sname VARCHAR(50),  rating INT,  age INT  );  CREATE TABLE Boats (  bid INT PRIMARY KEY,  bname VARCHAR(50),  color VARCHAR(20)  );  CREATE TABLE Reserves (  sid INT,  bid INT,  day DATE,  PRIMARY KEY (sid, bid, day),  FOREIGN KEY (sid) REFERENCES Sailors(sid),  FOREIGN KEY (bid) REFERENCES Boats(bid)  );  -- Step 4: Insert sample data into the tables  -- Insert data into Sailors  INSERT INTO Sailors (sid, sname, rating, age) VALUES  (1, 'Lubber', 5, 30),  (2, 'Penny', 7, 25),  (3, 'Sam', 4, 35),  (4, 'Tom', 6, 28);  -- Insert data into Boats  INSERT INTO Boats (bid, bname, color) VALUES  (101, 'Red Boat', 'red'),  (102, 'Blue Boat', 'blue'),  (103, 'Green Boat', 'green'),  (104, 'Yellow Boat', 'yellow'),  (105, 'Interlake', 'red'),  (106, 'Interlake', 'green');  -- Insert data into Reserves  INSERT INTO Reserves (sid, bid, day) VALUES  (1, 101, '2024-11-01'),  (1, 103, '2024-11-02'),  (2, 101, '2024-11-03'),  (3, 102, '2024-11-04'),  (3, 103, '2024-11-05'),  (4, 104, '2024-11-06'),  (2, 105, '2024-11-07'),  (3, 106, '2024-11-08');  -- Step 5: Run the queries  -- Query 1: Find the names of sailors who have reserved a red or a green boat  SELECT DISTINCT s.sname  FROM Sailors s  JOIN Reserves r ON s.sid = r.sid  JOIN Boats b ON r.bid = b.bid  WHERE b.color IN ('red', 'green');  -- Query 2: Find the names of sailors who have reserved both a red and a green boat  SELECT DISTINCT s.sname  FROM Sailors s  JOIN Reserves r1 ON s.sid = r1.sid  JOIN Boats b1 ON r1.bid = b1.bid  JOIN Reserves r2 ON s.sid = r2.sid  JOIN Boats b2 ON r2.bid = b2.bid  WHERE b1.color = 'red' AND b2.color = 'green';  -- Query 3: Find the names of sailors who have reserved at least two different boats  SELECT s.sname  FROM Sailors s  JOIN Reserves r ON s.sid = r.sid  GROUP BY s.sid  HAVING COUNT(DISTINCT r.bid) > 1;  -- Query 4: Find the sids of sailors with age over 20 who have not reserved a red boat  SELECT s.sid  FROM Sailors s  WHERE s.age > 20  AND s.sid NOT IN (  SELECT r.sid  FROM Reserves r  JOIN Boats b ON r.bid = b.bid  WHERE b.color = 'red'  );  -- Query 5: Find the names of sailors who have reserved all boats  SELECT s.sname  FROM Sailors s  WHERE NOT EXISTS (  SELECT b.bid  FROM Boats b  WHERE NOT EXISTS (  SELECT r.sid  FROM Reserves r  WHERE r.sid = s.sid AND r.bid = b.bid  )  );  -- Query 6: Find the names of sailors who have reserved all boats called Interlake  SELECT s.sname  FROM Sailors s  WHERE NOT EXISTS (  SELECT b.bid  FROM Boats b  WHERE b.bname = 'Interlake'  AND NOT EXISTS (  SELECT r.sid  FROM Reserves r  WHERE r.sid = s.sid AND r.bid = b.bid  )  );  -- Query 7: Find all sailors with a rating above 7  SELECT s.sname  FROM Sailors s  WHERE s.rating > 7;  -- Query 8: Find the sailor name, boat id, and reservation date for each reservation  SELECT s.sname, r.bid, r.day  FROM Reserves r  JOIN Sailors s ON r.sid = s.sid;  -- Query 9: Find sailors who have reserved all red boats  SELECT s.sname  FROM Sailors s  WHERE NOT EXISTS (  SELECT b.bid  FROM Boats b  WHERE b.color = 'red'  AND NOT EXISTS (  SELECT r.sid  FROM Reserves r  WHERE r.sid = s.sid AND r.bid = b.bid  )  );  -- Query 10: Find the names and ages of all sailors  SELECT s.sname, s.age  FROM Sailors s; |

|  |  |
| --- | --- |
| 32 | Flights(flno, from, to, distance, departs, arrives, price)  Aircraft(aid, aname, cruisingrange)  Certified(eid, aid)  Employees(eid, ename, salary)  Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly   1. Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots). 2. Print the name and salary of every nonpilot whose salary is more than the average salary for pilots. 3. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles. 4. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts. 5. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft. |
|  | -- Step 1: Create the database  CREATE DATABASE FlightDB;  -- Step 2: Use the database  USE FlightDB;  -- Step 3: Create tables  -- Table for Employees (including pilots and non-pilots)  CREATE TABLE Employees (  eid INT PRIMARY KEY,  ename VARCHAR(50),  salary DECIMAL(10,2)  );  -- Table for Aircraft  CREATE TABLE Aircraft (  aid INT PRIMARY KEY,  aname VARCHAR(50),  cruisingrange INT  );  -- Table for Flights  CREATE TABLE Flights (  flno INT PRIMARY KEY,  from\_city VARCHAR(50),  to\_city VARCHAR(50),  distance INT,  departs DATETIME,  arrives DATETIME,  price DECIMAL(10,2)  );  -- Table for Certified employees (pilots)  CREATE TABLE Certified (  eid INT,  aid INT,  PRIMARY KEY (eid, aid),  FOREIGN KEY (eid) REFERENCES Employees(eid),  FOREIGN KEY (aid) REFERENCES Aircraft(aid)  );  -- Step 4: Insert sample data into the tables  -- Insert data into Employees (some are pilots, some are not)  INSERT INTO Employees (eid, ename, salary) VALUES  (1, 'Alice', 50000),  (2, 'Bob', 60000),  (3, 'Charlie', 70000),  (4, 'David', 55000),  (5, 'Eva', 65000);  -- Insert data into Aircraft  INSERT INTO Aircraft (aid, aname, cruisingrange) VALUES  (101, 'Boeing 747', 12000),  (102, 'Boeing 777', 9000),  (103, 'Airbus A320', 5000),  (104, 'Boeing 787', 15000),  (105, 'Concorde', 4000);  -- Insert data into Certified (certifications of pilots)  INSERT INTO Certified (eid, aid) VALUES  (1, 101),  (2, 102),  (3, 103),  (4, 104),  (5, 101),  (2, 104);  -- Insert data into Flights  INSERT INTO Flights (flno, from\_city, to\_city, distance, departs, arrives, price) VALUES  (1001, 'New York', 'London', 3500, '2024-11-20 10:00:00', '2024-11-20 18:00:00', 1200),  (1002, 'Paris', 'Tokyo', 6000, '2024-11-21 12:00:00', '2024-11-21 20:00:00', 1500),  (1003, 'London', 'New York', 3500, '2024-11-22 14:00:00', '2024-11-22 22:00:00', 1100);  -- Step 5: Run the queries  -- Query 1: Compute the difference between the average salary of a pilot and the average salary of all employees  SELECT  (SELECT AVG(salary) FROM Employees WHERE eid IN (SELECT eid FROM Certified)) -  (SELECT AVG(salary) FROM Employees) AS salary\_difference;  -- Query 2: Print the name and salary of every non-pilot whose salary is more than the average salary for pilots  SELECT e.ename, e.salary  FROM Employees e  WHERE e.eid NOT IN (SELECT eid FROM Certified)  AND e.salary > (SELECT AVG(salary) FROM Employees WHERE eid IN (SELECT eid FROM Certified));  -- Query 3: Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles  SELECT e.ename  FROM Employees e  WHERE NOT EXISTS (  SELECT 1  FROM Certified c  JOIN Aircraft a ON c.aid = a.aid  WHERE c.eid = e.eid  AND a.cruisingrange <= 1000  )  AND e.eid IN (SELECT eid FROM Certified);  -- Query 4: Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts  SELECT e.ename  FROM Employees e  WHERE NOT EXISTS (  SELECT 1  FROM Certified c  JOIN Aircraft a ON c.aid = a.aid  WHERE c.eid = e.eid  AND a.cruisingrange <= 1000  )  AND e.eid IN (SELECT eid FROM Certified)  GROUP BY e.eid  HAVING COUNT(DISTINCT c.aid) >= 2;  -- Query 5: Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft  SELECT e.ename  FROM Employees e  WHERE NOT EXISTS (  SELECT 1  FROM Certified c  JOIN Aircraft a ON c.aid = a.aid  WHERE c.eid = e.eid  AND a.cruisingrange <= 1000  )  AND e.eid IN (SELECT eid FROM Certified)  AND EXISTS (  SELECT 1  FROM Certified c  JOIN Aircraft a ON c.aid = a.aid  WHERE c.eid = e.eid  AND a.aname LIKE '%Boeing%'  ); |

|  |  |
| --- | --- |
| 33 | Student(snum, sname, major, level, age)  Class(name, meets at, room, fid)  Enrolled(snum, cname)  Faculty(fid, fname, deptid)  The meaning of these relations is straightforward; for example, Enrolled has one record  per student-class pair such that the student is enrolled in the class.  Find the names of all juniors (Level = JR) who are enrolled in a class taught by I. Teacher.  2. Find the age of the oldest student who is either a History major or is enrolled in a course taught by I. Teacher.  3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.  4. Find the names of all students who are enrolled in two classes that meet at the same time.  5. Find the names of faculty members who teach in every room in which some class is taught.  6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.  7. Print the Level and the average age of students for that Level, for each Level.  8. Print the Level and the average age of students for that Level, for all Levels except JR.  9. Find the names of students who are enrolled in the maximum number of classes.  10. Find the names of students who are not enrolled in any class. |
|  | -- Step 1: Create the database  CREATE DATABASE UniversityDB;  -- Step 2: Use the database  USE UniversityDB;  -- Step 3: Create tables  -- Table for Students  CREATE TABLE Student (  snum INT PRIMARY KEY,  sname VARCHAR(50),  major VARCHAR(50),  level VARCHAR(10), -- Levels can be 'FR' for Freshman, 'SO' for Sophomore, 'JR' for Junior, 'SR' for Senior  age INT  );  -- Table for Classes  CREATE TABLE Class (  name VARCHAR(50) PRIMARY KEY,  meets\_at DATETIME,  room VARCHAR(50),  fid INT  );  -- Table for Enrollments  CREATE TABLE Enrolled (  snum INT,  cname VARCHAR(50),  PRIMARY KEY (snum, cname),  FOREIGN KEY (snum) REFERENCES Student(snum),  FOREIGN KEY (cname) REFERENCES Class(name)  );  -- Table for Faculty  CREATE TABLE Faculty (  fid INT PRIMARY KEY,  fname VARCHAR(50),  deptid INT  );  -- Step 4: Insert sample data into the tables  -- Insert students  INSERT INTO Student (snum, sname, major, level, age) VALUES  (1, 'Alice', 'History', 'JR', 20),  (2, 'Bob', 'Computer Science', 'SO', 19),  (3, 'Charlie', 'History', 'SR', 22),  (4, 'David', 'Math', 'JR', 21),  (5, 'Eve', 'History', 'FR', 18);  -- Insert faculty  INSERT INTO Faculty (fid, fname, deptid) VALUES  (1, 'I. Teacher', 101),  (2, 'John Smith', 102),  (3, 'Mary Johnson', 103);  -- Insert classes  INSERT INTO Class (name, meets\_at, room, fid) VALUES  ('History101', '2024-11-20 09:00:00', 'R128', 1),  ('Math101', '2024-11-20 09:00:00', 'R129', 2),  ('History201', '2024-11-21 10:00:00', 'R128', 1),  ('Math202', '2024-11-21 10:00:00', 'R130', 3),  ('CS101', '2024-11-22 11:00:00', 'R129', 2);  -- Insert enrollments  INSERT INTO Enrolled (snum, cname) VALUES  (1, 'History101'),  (1, 'History201'),  (2, 'Math101'),  (3, 'History101'),  (4, 'Math202'),  (5, 'CS101'),  (5, 'Math101');  -- Step 5: Run the queries  -- Query 1: Find the names of all juniors (Level = JR) who are enrolled in a class taught by 'I. Teacher'.  SELECT s.sname  FROM Student s  JOIN Enrolled e ON s.snum = e.snum  JOIN Class c ON e.cname = c.name  JOIN Faculty f ON c.fid = f.fid  WHERE s.level = 'JR' AND f.fname = 'I. Teacher';  -- Query 2: Find the age of the oldest student who is either a History major or is enrolled in a course taught by 'I. Teacher'.  SELECT MAX(s.age)  FROM Student s  WHERE s.major = 'History'  UNION  SELECT MAX(s.age)  FROM Student s  JOIN Enrolled e ON s.snum = e.snum  JOIN Class c ON e.cname = c.name  JOIN Faculty f ON c.fid = f.fid  WHERE f.fname = 'I. Teacher';  -- Query 3: Find the names of all classes that either meet in room R128 or have five or more students enrolled.  SELECT c.name  FROM Class c  LEFT JOIN Enrolled e ON c.name = e.cname  GROUP BY c.name  HAVING c.room = 'R128' OR COUNT(e.snum) >= 5;  -- Query 4: Find the names of all students who are enrolled in two classes that meet at the same time.  SELECT s.sname  FROM Student s  JOIN Enrolled e1 ON s.snum = e1.snum  JOIN Enrolled e2 ON s.snum = e2.snum AND e1.cname <> e2.cname  JOIN Class c1 ON e1.cname = c1.name  JOIN Class c2 ON e2.cname = c2.name  WHERE c1.meets\_at = c2.meets\_at;  -- Query 5: Find the names of faculty members who teach in every room in which some class is taught.  SELECT f.fname  FROM Faculty f  WHERE NOT EXISTS (  SELECT 1  FROM Class c  WHERE c.fid = f.fid  AND NOT EXISTS (  SELECT 1  FROM Class c2  WHERE c2.room = c.room  )  );  -- Query 6: Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.  SELECT f.fname  FROM Faculty f  JOIN Class c ON f.fid = c.fid  LEFT JOIN Enrolled e ON c.name = e.cname  GROUP BY f.fid  HAVING COUNT(e.snum) < 5;  -- Query 7: Print the Level and the average age of students for that Level, for each Level.  SELECT s.level, AVG(s.age) AS avg\_age  FROM Student s  GROUP BY s.level;  -- Query 8: Print the Level and the average age of students for that Level, for all Levels except JR.  SELECT s.level, AVG(s.age) AS avg\_age  FROM Student s  WHERE s.level != 'JR'  GROUP BY s.level;  -- Query 9: Find the names of students who are enrolled in the maximum number of classes.  SELECT s.sname  FROM Student s  JOIN Enrolled e ON s.snum = e.snum  GROUP BY s.snum  HAVING COUNT(e.cname) = (  SELECT MAX(class\_count)  FROM (  SELECT COUNT(cname) AS class\_count  FROM Enrolled  GROUP BY snum  ) AS subquery  );  -- Query 10: Find the names of students who are not enrolled in any class.  SELECT s.sname  FROM Student s  LEFT JOIN Enrolled e ON s.snum = e.snum  WHERE e.snum IS NULL; |

|  |  |
| --- | --- |
| 33 | 1)DEPT Table  deptno number(2,0),  dname varchar2(14),  loc varchar2(13),  Primary key deptno  2)EMP Table  empno number(4,0),  ename varchar2(10),  job varchar2(9),  mgr number(4,0),  hiredate date,  sal number(7,2),  comm number(7,2),  deptno number(2,0),  Pk empno  Fk deptno  **QUERIES**   1. List all the employees who have at least one person reporting to them. 2. List the employee details if and only if more than 5 employees are present in department no 10 3. List the name of the employees with their immediate higher authority. 4. List all the employees who do not manage any one. 5. List the employee details whose salary is greater than the lowest salary of an employee belonging to deptno 20. 6. List the details of the employee earning more than the highest paid manager. 7. List the highest salary paid for each job. 8. Find the most recently hired employee in each department. 9. In which year did most people join the company? Display the year and the number of employees. 10. Which department has the highest annual remuneration bill? 11. Write a query to display a ‘\*’ against the row of the most recently hired employee. 12. Write a correlated sub-query to list out the employees who earn more than the average salary of their department. 13. Find the nth maximum salary. 14. Select the duplicate records (Records, which are inserted, that already exist) in the EMP table. 15. Write a query to list the length of service of the employees (of the form n years and m months). |
|  | -- Creating the DEPT table  CREATE TABLE DEPT (  deptno INT(2) PRIMARY KEY,  dname VARCHAR(14),  loc VARCHAR(13)  );  -- Creating the EMP table  CREATE TABLE EMP (  empno INT(4) PRIMARY KEY,  ename VARCHAR(10),  job VARCHAR(9),  mgr INT(4), -- Manager's empno  hiredate DATE,  sal DECIMAL(7,2),  comm DECIMAL(7,2),  deptno INT(2),  FOREIGN KEY (deptno) REFERENCES DEPT(deptno),  FOREIGN KEY (mgr) REFERENCES EMP(empno) -- Self-reference for the manager  );  -- Inserting sample data into DEPT table  INSERT INTO DEPT (deptno, dname, loc) VALUES (10, 'ACCOUNTING', 'NEW YORK');  INSERT INTO DEPT (deptno, dname, loc) VALUES (20, 'RESEARCH', 'DALLAS');  INSERT INTO DEPT (deptno, dname, loc) VALUES (30, 'SALES', 'CHICAGO');  INSERT INTO DEPT (deptno, dname, loc) VALUES (40, 'OPERATIONS', 'BOSTON');  -- Inserting sample data into EMP table  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7839, 'KING', 'PRESIDENT', NULL, '1990-06-09', 5000, NULL, 10);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7566, 'JONES', 'MANAGER', 7839, '1992-01-10', 2975, NULL, 20);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7698, 'BLAKE', 'MANAGER', 7839, '1991-05-14', 2850, NULL, 30);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7782, 'CLARK', 'MANAGER', 7839, '1992-05-14', 2450, NULL, 10);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7788, 'SCOTT', 'ANALYST', 7566, '1993-06-13', 3000, NULL, 20);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7902, 'FORD', 'ANALYST', 7566, '1994-12-05', 3000, NULL, 20);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7844, 'TURNER', 'SALESMAN', 7698, '1992-01-21', 1500, 0.3, 30);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7900, 'JAMES', 'CLERK', 7698, '2000-01-01', 950, NULL, 30);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7654, 'MARTIN', 'SALESMAN', 7698, '1998-10-11', 1250, 0.2, 30);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7934, 'MILLER', 'CLERK', 7782, '2001-01-21', 1300, NULL, 10);  INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)  VALUES (7935, 'SIMON', 'CLERK', 7782, '2022-03-23', 1100, NULL, 10);  -- 1) List all the employees who have at least one person reporting to them  SELECT DISTINCT e.ename  FROM EMP e  WHERE e.empno IN (SELECT mgr FROM EMP WHERE mgr IS NOT NULL);  -- 2) List the employee details if and only if more than 5 employees are present in department no 10  SELECT \*  FROM EMP e  WHERE e.deptno = 10  AND (SELECT COUNT(\*) FROM EMP WHERE deptno = 10) > 5;  -- 3) List the name of the employees with their immediate higher authority  SELECT e1.ename AS employee\_name, e2.ename AS manager\_name  FROM EMP e1  LEFT JOIN EMP e2 ON e1.mgr = e2.empno;  -- 4) List all the employees who do not manage anyone  SELECT e.ename  FROM EMP e  WHERE e.empno NOT IN (SELECT DISTINCT mgr FROM EMP WHERE mgr IS NOT NULL);  -- 5) List the employee details whose salary is greater than the lowest salary of an employee belonging to deptno 20  SELECT \*  FROM EMP  WHERE sal > (SELECT MIN(sal) FROM EMP WHERE deptno = 20);  -- 6) List the details of the employee earning more than the highest paid manager  SELECT \*  FROM EMP  WHERE sal > (SELECT MAX(sal) FROM EMP WHERE job = 'MANAGER');  -- 7) List the highest salary paid for each job  SELECT job, MAX(sal) AS highest\_salary  FROM EMP  GROUP BY job;  -- 8) Find the most recently hired employee in each department  SELECT deptno, ename, hiredate  FROM EMP  WHERE (deptno, hiredate) IN (SELECT deptno, MAX(hiredate) FROM EMP GROUP BY deptno);  -- 9) In which year did most people join the company? Display the year and the number of employees  SELECT YEAR(hiredate) AS year, COUNT(\*) AS num\_employees  FROM EMP  GROUP BY YEAR(hiredate)  ORDER BY num\_employees DESC  LIMIT 1;  -- 10) Which department has the highest annual remuneration bill?  SELECT deptno, SUM(sal) AS total\_remuneration  FROM EMP  GROUP BY deptno  ORDER BY total\_remuneration DESC  LIMIT 1;  -- 11) Write a query to display a ‘\*’ against the row of the most recently hired employee  SELECT ename, hiredate,  CASE WHEN hiredate = (SELECT MAX(hiredate) FROM EMP) THEN '\*' ELSE '' END AS recently\_hired  FROM EMP;  -- 12) Write a correlated sub-query to list out the employees who earn more than the average salary of their department  SELECT ename, sal, deptno  FROM EMP e  WHERE sal > (SELECT AVG(sal) FROM EMP WHERE deptno = e.deptno);  -- 13) Find the nth maximum salary  -- Replace 'n' with the desired rank (e.g., 2nd, 3rd max salary)  SELECT MAX(sal) AS nth\_max\_salary  FROM EMP  WHERE sal NOT IN (SELECT sal FROM EMP ORDER BY sal DESC LIMIT n-1);  -- 14) Select the duplicate records (Records, which are inserted, that already exist) in the EMP table.  SELECT empno, ename, job, hiredate, sal, comm, deptno  FROM EMP  GROUP BY empno, ename, job, hiredate, sal, comm, deptno  HAVING COUNT(\*) > 1;  -- 15) Write a query to list the length of service of the employees (of the form n years and m months)  SELECT ename,  FLOOR(TIMESTAMPDIFF(MONTH, hiredate, CURDATE()) / 12) AS years,  MOD(TIMESTAMPDIFF(MONTH, hiredate, CURDATE()), 12) AS months  FROM EMP; |