1) To study DDL-create and DML-insert commands . (i)Create tables according to the following definition. CREATETABLEDEPOSIT(ACTNOVARCHAR2(5), CNAMEVARCHAR2(18), BNAMEVARCHAR2(18), AMOUNTNUMBER(8,2) ,ADATEDATE); CREATETABLEBRANCH(BNAMEVARCHAR2(18),CITY VARCHAR2(18)); CREATETABLECUSTOMERS(CNAMEVARCHAR2(19),CITYVARCHAR2(18)); CREATETABLEBORROW(LOANNOVARCHAR2(5),CNAMEVARCHAR2(18), BNAMEVARCHAR2(18), AMOUNTNUMBER(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'

Ans:

CREATE TABLE DEPOSIT (ACTNO VARCHAR(5), CNAME VARCHAR(18),BNAME VARCHAR(18),AMOUNT DECIMAL(8, 2),ADATE DATE);

CREATE TABLE BRANCH (BNAME VARCHAR(18),CITY VARCHAR(18));

CREATE TABLE CUSTOMERS (CNAME VARCHAR(19), CITY VARCHAR(18));

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', 'Central', 5000.00, '1996-12-15'),

('A002', 'Jane Smith', 'North', 3000.00, '1996-11-20'),

('A003', 'Robert Brown', 'East', 7000.00, '1997-01-10');

-- Insert data into BRANCH

INSERT INTO BRANCH (BNAME, CITY)

VALUES

('Central', 'New York'),

('North', 'Chicago'),

('East', 'Los Angeles');

-- Insert data into CUSTOMERS

INSERT INTO CUSTOMERS (CNAME, CITY)

VALUES

```
('John Doe', 'New York'),
('Jane Smith', 'Chicago'),
('Robert Brown', 'Los Angeles');

-- Insert data into BORROW
INSERT INTO BORROW (LOANNO, CNAME, BNAME, AMOUNT)
VALUES
('L001', 'John Doe', 'Central', 8000.00),
('L002', 'Jane Smith', 'North', 4000.00),
('L003', 'Robert Brown', 'East', 10000.00);
DESCRIBE DEPOSIT;
DESCRIBE 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';
```

2) 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 '

Ans:

```
1)SELECT * FROM Employee;

SELECT * FROM Job;

SELECT * FROM Deposit;

2)SELECT a_no, amount

FROM Deposit

WHERE a_date BETWEEN '2006-01-01' AND '2006-07-25';

3)SELECT *

FROM Job

WHERE min_sal > 4000;

4)SELECT emp_name AS EmployeeName, emp_sal AS Salary

FROM Employee

WHERE dept_no = 20;

5)SELECT emp_no, emp_name, dept_no

FROM Employee

WHERE dept_no IN (10, 20);

1)SELECT *

FROM Employee

WHERE emp_name LIKE 'A_a%';

2)SELECT emp_name, emp_no, emp_sal

FROM Employee

WHERE emp_name LIKE 'Ani__';

3)SELECT emp_name, emp_no, emp_sal

FROM Employee

WHERE emp_name LIKE '_n___'

  AND emp_name IS NOT NULL;

4)SELECT emp_name

FROM Employee

WHERE emp_name IS NULL

  AND emp_name LIKE '__a%';

5)SELECT emp_name

FROM Employee
```

WHERE emp_name LIKE '%_%' ESCAPE '\';

3) 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

Ans:

1) SELECT SUM(amount) AS Total_Deposit

FROM Deposit;

2) SELECT SUM(amount) AS Total_Loan

FROM Borrow

WHERE bname = 'Karolbagh';

3) SELECT MAX(amount) AS Max_Loan

FROM Borrow

WHERE bname = 'VRCE';

4) SELECT COUNT(DISTINCT cname) AS Total_Customers

FROM Deposit;

5) SELECT COUNT(DISTINCT bname) AS Total_Cities

FROM Deposit;

6) CREATE TABLE supplier AS

SELECT * FROM Employee;

7) CREATE TABLE sup1 AS

SELECT emp_no, emp_name

FROM Employee;

8) CREATE TABLE sup2 AS

SELECT * FROM Employee WHERE 1 = 0;

9) INSERT INTO sup2

```
SELECT *
FROM Employee
WHERE emp_name LIKE '_n___';
```

10) DELETE FROM sup1;

```
11) DELETE FROM supplier
WHERE emp_no = 103;
```

12) RENAME TABLE sup2 TO renamed_sup2;

13) DROP TABLE sup1;

```
14) UPDATE Employee
SET dept_no = 10
WHERE emp_name LIKE '_m%';
```

```
15) UPDATE Employee
SET emp_name = 'Updated Name'
WHERE emp_no = 103;
```

4) 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.).

Ans:

1) SELECT CURDATE() AS Date;

```
2) SELECT
   emp_no AS "Employee Number",
   emp_job AS "Job",
   emp_sal AS "Salary",
   ROUND(emp_sal * 1.15) AS "New Salary"
```

```
FROM Employee;
3) SELECT
   emp_no AS "Employee Number",
   emp_job AS "Job",
   emp_sal AS "Salary",
   ROUND(emp_sal * 1.15) AS "New Salary",
   ROUND(emp_sal * 1.15) - emp_sal AS "Increase"
FROM Employee;
4) 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) SELECT
   emp_name AS "Employee Name",
   ROUND(emp_sal / 12, 2) AS "Monthly Earnings"
FROM Employee;
6) 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) SELECT
   DATE_FORMAT(hire_date, '%D of %M %Y %r') AS "Formatted Hire Date"
FROM Employee;
8) SELECT
   emp_name AS "Employee Name",
```

```
   (emp_sal + IFNULL(emp_comm, 0)) * 12 AS "Annual Compensation"
FROM Employee;
```

5) 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

ANS:

- **Customer**: Customer(cname, city, bname)
- **Borrow**: Borrow(loanno, cname, bname, amount)
- **Deposit**: Deposit(a_no, cname, bname, amount, a_date)
- **Employee**: Employee(emp_no, emp_name, emp_lname, dept_no, hire_date, mgr_no)
- **Department**: Department(dept_no, dept_name, location)

```
1) SELECT *
FROM Customer
WHERE cname = 'ANIL';
2) SELECT DISTINCT b.cname
FROM Borrow b
JOIN Deposit d ON b.cname = d.cname
JOIN Customer c ON c.cname = b.cname
WHERE c.city = 'Nagpur';
3) SELECT c1.city AS "Their City Name"
FROM Customer c1
JOIN Customer c2 ON c1.bname = c2.bname
WHERE c1.city != c2.city;
4) SELECT e.emp_lname AS "Last Name",
    e.dept_no AS "Department Number",
    d.dept_name AS "Department Name"
FROM Employee e
```

```
JOIN Department d ON e.dept_no = d.dept_no;
5) SELECT DISTINCT e.emp_job AS "Job Title",
    d.location AS "Location"
FROM Employee e
JOIN Department d ON e.dept_no = d.dept_no
WHERE e.dept_no = 30;
6) SELECT e.emp_name AS "Employee Name",
    e.dept_no AS "Department Number",
    d.dept_name AS "Department Name"
FROM Employee e
JOIN Department d ON e.dept_no = d.dept_no
WHERE d.location = 'NEW YORK';
7) SELECT e.emp_lname AS "Employee",
    e.emp_no AS "Emp#",
    m.emp_lname AS "Manager",
    m.emp_no AS "Mgr#"
FROM Employee e
LEFT JOIN Employee m ON e.mgr_no = m.emp_no;
8) SELECT e.emp_name AS "Employee Name",
    e.hire_date AS "Hire Date"
FROM Employee e
JOIN Employee scott ON scott.emp_name = 'SCOTT'
WHERE e.hire_date > scott.hire_date;
```

6) 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.

Ans:

1) SELECT CustomerID, SUM(Deposit) AS TotalDeposit

FROM Accounts

WHERE AccountDate > TO_DATE('1996-01-01', 'YYYY-MM-DD')

GROUP BY CustomerID;

2) SELECT CustomerID, SUM(Deposit) AS TotalDeposit

FROM Accounts A

JOIN Customers C ON A.CustomerID = C.CustomerID

WHERE C.City = 'Nagpur'

GROUP BY CustomerID;

3) SELECT CustomerID, MAX(Deposit) AS MaxDeposit

FROM Accounts A

JOIN Customers C ON A.CustomerID = C.CustomerID

WHERE C.City = 'Bombay'

GROUP BY CustomerID;

4) SELECT

ROUND(MAX(Salary)) AS Maximum,

ROUND(MIN(Salary)) AS Minimum,

ROUND(SUM(Salary)) AS Sum,

ROUND(AVG(Salary)) AS Average

FROM Employees;

5) SELECT

(MAX(Salary) - MIN(Salary)) AS DIFFERENCE

FROM Employees;

6) SELECT

COUNT(*) AS TotalEmployees,

SUM(CASE WHEN EXTRACT(YEAR FROM HireDate) = 1995 THEN 1 ELSE 0 END) AS Hired1995,

SUM(CASE WHEN EXTRACT(YEAR FROM HireDate) = 1996 THEN 1 ELSE 0 END) AS Hired1996,

```
   SUM(CASE WHEN EXTRACT(YEAR FROM HireDate) = 1997 THEN 1 ELSE 0 END) AS Hired1997,
   SUM(CASE WHEN EXTRACT(YEAR FROM HireDate) = 1998 THEN
```

```
7) SELECT AVG(Salary) AS AverageSalary
FROM Employees
GROUP BY DepartmentID;
```

```
8) SELECT DepartmentID, JobTitle, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY DepartmentID, JobTitle;
```

```
9) SELECT AVG(Salary) AS AverageSalary
FROM Employees
WHERE Salary > 2000
GROUP BY DepartmentID;
```

```
10) SELECT JobTitle, SUM(Salary) AS TotalSalary
FROM Employees
WHERE JobTitle <> 'President'
GROUP BY JobTitle
HAVING SUM(Salary) > 3000
ORDER BY TotalSalary DESC;
```

```
11) SELECT BranchID, SUM(Deposit) AS TotalDeposit
FROM Branches B
JOIN Accounts A ON B.BranchID = A.BranchID
WHERE B.City = 'Bombay'
GROUP BY BranchID
HAVING SUM(Deposit) > 5000;
```

7) 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.

Ans:

```
1) SELECT LastName, HireDate
FROM Employees
WHERE DepartmentID = (
   SELECT DepartmentID
   FROM Employees
   WHERE LastName = 'SCOTT'
)
AND LastName != 'SCOTT';
2) SELECT CustomerName
FROM Customers
WHERE BranchCity = (
   SELECT BranchCity
   FROM Customers
   WHERE CustomerName = 'Sunil'
);
3) SELECT *
FROM Deposits
WHERE CustomerID IN (
   SELECT CustomerID
   FROM Customers
   WHERE City = (
      SELECT City
      FROM Customers
      WHERE CustomerName = 'Pramod'
   )
)
UNION
SELECT *
```

```
FROM Loans
WHERE CustomerID IN (
   SELECT CustomerID
   FROM Customers
   WHERE City = (
      SELECT City
      FROM Customers
      WHERE CustomerName = 'Pramod'
   )
);
4) SELECT EmployeeID, LastName
FROM Employees
WHERE Salary > (
   SELECT AVG(Salary)
   FROM Employees
)
ORDER BY Salary ASC;
5) SELECT CustomerName
FROM Customers
WHERE City = (
   SELECT City
   FROM Customers
   WHERE CustomerName = 'Anil'
)
AND CustomerID IN (
   SELECT CustomerID
   FROM Deposits
   WHERE DepositAmount > 2000
);
6) SELECT LastName, Salary
FROM Employees
```

```
WHERE ManagerID = (
   SELECT EmployeeID
   FROM Employees
   WHERE LastName = 'Ford'
);
7) SELECT DepartmentID, LastName, JobTitle
FROM Employees
WHERE DepartmentID = (
   SELECT DepartmentID
   FROM Departments
   WHERE DepartmentName = 'Accounting'
);
8) SELECT BranchName
FROM Branches
WHERE BranchID = (
   SELECT BranchID
   FROM Deposits
   GROUP BY BranchID
   ORDER BY COUNT(CustomerID) DESC
   LIMIT 1
);
9) SELECT City
FROM Branches
GROUP BY City
HAVING COUNT(BranchID) = (
   SELECT MAX(BranchCount)
   FROM (
      SELECT COUNT(BranchID) AS BranchCount
      FROM Branches
      GROUP BY City
   ) SubQuery
```

```
);
10) SELECT CustomerName
FROM Customers
WHERE City = (
  SELECT City
  FROM Customers
  WHERE CustomerID IN (
    SELECT CustomerID
    FROM Deposits
  )
  GROUP BY City
  ORDER BY COUNT(CustomerID) DESC
  LIMIT 1
);
```

8) 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)Deleteborrowerofbrancheshavingaverageloanlessthan1000

Ans:

```
1) UPDATE Deposits
SET DepositAmount = DepositAmount * 1.10;
2) UPDATE Deposits
SET DepositAmount = DepositAmount * 1.10
WHERE BranchID = (
  SELECT BranchID
  FROM Branches
  WHERE BranchName = 'vrce'
);
```

```
3) UPDATE Deposits
SET DepositAmount = DepositAmount * 1.10
WHERE CustomerID IN (
   SELECT CustomerID
   FROM Customers
   WHERE City = 'Nagpur'
)
AND BranchID IN (
   SELECT BranchID
   FROM Branches
   WHERE City = 'Bombay'
);
4) UPDATE Employees
SET DepartmentID = (
   SELECT DepartmentID
   FROM Employees
   WHERE EmployeeID = 7844
)
WHERE JobTitle = (
   SELECT JobTitle
   FROM Employees
   WHERE EmployeeID = 7788
);
5) UPDATE Employees
SET DepartmentID = 10 -- Replace `10` with the desired department ID
WHERE EmployeeID = 1234; -- Replace `1234` with the specific employee number
6) UPDATE Deposits
SET DepositAmount = DepositAmount - 10
WHERE CustomerID = (
   SELECT CustomerID
   FROM Customers
```

```
   WHERE CustomerName = 'Anil'
)
AND BranchID = (
   SELECT BranchID
   FROM Customers
   WHERE CustomerName = 'Sunil'
);

UPDATE Deposits
SET DepositAmount = DepositAmount + 10
WHERE CustomerID = (
   SELECT CustomerID
   FROM Customers
   WHERE CustomerName = 'Sunil'
)
AND BranchID = (
   SELECT BranchID
   FROM Customers
   WHERE CustomerName = 'Anil'
);
7) UPDATE Deposits
SET DepositAmount = DepositAmount + 100
WHERE CustomerID IN (
   SELECT CustomerID
   FROM (
      SELECT CustomerID, MAX(DepositAmount) AS MaxDeposit
      FROM Deposits
      GROUP BY BranchID
   ) SubQuery
   WHERE DepositAmount = MaxDeposit
);
```

```
8) DELETE FROM Deposits
WHERE CustomerID = (
   SELECT CustomerID
   FROM Customers
   WHERE CustomerName = 'Vijay'
);
9) DELETE FROM Borrowers
WHERE BranchID IN (
   SELECT BranchID
   FROM Loans
   GROUP BY BranchID
   HAVING AVG(LoanAmount) < 1000
);
```

9) ToPerformOperationsUsingPL/SQL. 1.WriteaPL/SQLBlockthatwillget thesalaryofemployeewithemployeenumber '105'anddisplayitontheScreen 2.WriteaPL/SQLblockthatdemonstratesuseofCONSTANT. 3.WriteaPL/SQLBlockthatdemonstratesDecisionmakingStatements. 4.Write a PL/SQLblock that prints 1 to 5numbersUsingLOOP…EXITWHEN Statement. 5.WriteaPL/SQLblockthatprints1to5numbersUsingWHILELoopStatement

Ans:

```
1) DECLARE
   v_salary NUMBER; -- Variable to store the salary
BEGIN
   SELECT Salary
   INTO v_salary
   FROM Employees
   WHERE EmployeeID = 105;

   DBMS_OUTPUT.PUT_LINE('The salary of employee 105 is: ' || v_salary);
END;
/
2) DECLARE
```

```
   v_constant_value CONSTANT NUMBER := 100; -- Define a constant
   v_result NUMBER;
BEGIN
   v_result := v_constant_value * 2;
   DBMS_OUTPUT.PUT_LINE('The result is: ' || v_result);
END;
/
3) DECLARE
   v_salary NUMBER := 5000; -- Example salary
BEGIN
   IF v_salary > 10000 THEN
      DBMS_OUTPUT.PUT_LINE('Salary is above 10,000.');
   ELSIF v_salary BETWEEN 5000 AND 10000 THEN
      DBMS_OUTPUT.PUT_LINE('Salary is between 5,000 and 10,000.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Salary is below 5,000.');
   END IF;
END;
/
4) DECLARE
   v_counter NUMBER := 1; -- Counter variable
BEGIN
   LOOP
      DBMS_OUTPUT.PUT_LINE(v_counter);
      v_counter := v_counter + 1;

      EXIT WHEN v_counter > 5; -- Exit condition
   END LOOP;
END;
/
5) DECLARE
```

```
  v_counter NUMBER := 1; -- Counter variable
BEGIN
  WHILE v_counter <= 5 LOOP
    DBMS_OUTPUT.PUT_LINE(v_counter);
    v_counter := v_counter + 1;
  END LOOP;
END;
/
```

10) 6.WriteaPL/SQLblockthatprints1to5numbersUsingFORLoopStatement. 7.Write a PL/SQLblock that demonstratesUseofSQLStatements insidePL/SQL BLOCK
8.WriteaPL/SQLblockthatimplementsImplicitCursor.
9.WriteaPL/SQLblockthatimplementsExplicitCursor. 10.WriteaPL/SQLblock that implementsStoredProcedurewithIN,OUT, INOUT ParameterswithEXPEPTIONHANDLINGmechanism

Ans:

```
6) BEGIN
  FOR v_counter IN 1..5 LOOP
    DBMS_OUTPUT.PUT_LINE(v_counter);
  END LOOP;
END;
/
7) DECLARE
  v_total_salary NUMBER;
BEGIN
  SELECT SUM(Salary)
  INTO v_total_salary
  FROM Employees;

  DBMS_OUTPUT.PUT_LINE('The total salary of all employees is: ' || v_total_salary);
END;
/
8) DECLARE
```

```
    v_employee_name VARCHAR2(50);
    v_salary NUMBER;
BEGIN
    SELECT EmployeeName, Salary
    INTO v_employee_name, v_salary
    FROM Employees
    WHERE EmployeeID = 105;

    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_employee_name || ', Salary: ' || v_salary);
END;
/
```

9)

```
DECLARE
    CURSOR emp_cursor IS
        SELECT EmployeeID, EmployeeName, Salary
        FROM Employees;
    v_employee_id NUMBER;
    v_employee_name VARCHAR2(50);
    v_salary NUMBER;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_employee_id, v_employee_name, v_salary;
        EXIT WHEN emp_cursor%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id ||
                             ', Name: ' || v_employee_name ||
                             ', Salary: ' || v_salary);
    END LOOP;
    CLOSE emp_cursor;
END;
/
```

10) CREATE OR REPLACE PROCEDURE ManageSalary (

```
    p_employee_id IN NUMBER,
    p_bonus INOUT NUMBER,
    p_final_salary OUT NUMBER
) AS
    v_current_salary NUMBER;
BEGIN
    -- Retrieve the current salary
    SELECT Salary
    INTO v_current_salary
    FROM Employees
    WHERE EmployeeID = p_employee_id;

    -- Calculate the final salary
    p_bonus := p_bonus + 500; -- Update the bonus
    p_final_salary := v_current_salary + p_bonus;

    -- Output the result
    DBMS_OUTPUT.PUT_LINE('Final Salary for Employee ' || p_employee_id || ' is: ' || p_final_salary);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employee found with ID ' || p_employee_id);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/

-- Example of calling the procedure
DECLARE
    v_bonus NUMBER := 1000;
    v_final_salary NUMBER;
```

```
BEGIN
   ManageSalary(105, v_bonus, v_final_salary);
   DBMS_OUTPUT.PUT_LINE('Updated Bonus: ' || v_bonus);
   DBMS_OUTPUT.PUT_LINE('Final Salary: ' || v_final_salary);
END;
/
```

11) 11.WriteaPL/SQLblockthatimplementsFunction.
12.WriteaPL/SQLblockthatimplementsAFTERUPDATETRIGGER.
13.WriteaPL/SQLblockthatimplementsBEFOREUPDATETRIGGER
14.Writeatriggertocheckthemarkisnot ZeroorNegative.
15.WriteaTriggerthatchecktheemployeenamemuststartswith'M'

Ans:

```
11) CREATE OR REPLACE FUNCTION CalculateBonus(p_salary NUMBER)
RETURN NUMBER
AS
   v_bonus NUMBER;
BEGIN
   IF p_salary >= 10000 THEN
      v_bonus := p_salary * 0.10; -- 10% bonus
   ELSE
      v_bonus := p_salary * 0.05; -- 5% bonus
   END IF;

   RETURN v_bonus;
END;
/

12) CREATE OR REPLACE TRIGGER AfterSalaryUpdate
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
   INSERT INTO SalaryLogs(EmployeeID, OldSalary, NewSalary, UpdateDate)
```

```
    VALUES (:OLD.EmployeeID, :OLD.Salary, :NEW.Salary, SYSDATE);

    DBMS_OUTPUT.PUT_LINE('Salary updated for Employee ID: ' || :NEW.EmployeeID);
END;
/
13) CREATE OR REPLACE TRIGGER BeforeSalaryUpdate
BEFORE UPDATE ON Employees
FOR EACH ROW
BEGIN
    IF :NEW.Salary < :OLD.Salary THEN
        RAISE_APPLICATION_ERROR(-20001, 'Salary cannot be decreased.');
    END IF;

    DBMS_OUTPUT.PUT_LINE('Salary update validated for Employee ID: ' || :NEW.EmployeeID);
END;
/
14) CREATE OR REPLACE TRIGGER CheckMarks
BEFORE INSERT OR UPDATE ON Students
FOR EACH ROW
BEGIN
    IF :NEW.Marks <= 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Marks cannot be zero or negative.');
    END IF;
END;
/
15) CREATE OR REPLACE TRIGGER CheckEmployeeName
BEFORE INSERT OR UPDATE ON Employees
FOR EACH ROW
BEGIN
    IF SUBSTR(:NEW.EmployeeName, 1, 1) != 'M' THEN
        RAISE_APPLICATION_ERROR(-20003, 'Employee name must start with "M".');
```

```
    END IF;
END;
/
```

12) Considerthefollowingrelations:
Student(snum:integer,sname:string,major:string,level:string,age:integer)
Class(name:string,meetsat:string,room:string,fid:integer) Enrolled(snum:integer,cname:string)
Faculty(fid:integer,fname:string,deptid:integer)
Themeaningoftheserelationsisstraightforward;forexample,Enrolledhasonerecord perstudent-classpairsuchthatthestudentisenrolledintheclass.
WritethefollowingqueriesinSQL.Noduplicatesshouldbeprintedinanyofthe 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 SOstudents aged 18, you should print the pair (18, FR).

Ans:

1)
```
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)
```
SELECT MAX(S.age)
FROM Student S
WHERE S.major = 'History'
  OR S.snum IN (
      SELECT E.snum
      FROM Enrolled E
      JOIN Class C ON E.cname = C.name
```

```
      JOIN Faculty F ON C.fid = F.fid
      WHERE F.fname = 'I. Teach'
   );
3) SELECT DISTINCT C.name
FROM Class C
WHERE C.room = 'R128'
  OR C.name IN (
      SELECT E.cname
      FROM Enrolled E
      GROUP BY E.cname
      HAVING COUNT(E.snum) >= 5
   );
4) 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 E1.cname <> E2.cname AND C1.meetsat = C2.meetsat;
5) SELECT DISTINCT F.fname
FROM Faculty F
WHERE NOT EXISTS (
      SELECT DISTINCT C.room
      FROM Class C
      WHERE C.room NOT IN (
         SELECT C1.room
         FROM Class C1
         WHERE C1.fid = F.fid
      )
   );
6) SELECT DISTINCT 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.fname
HAVING COUNT(E.snum) < 5;
```

7)
```
SELECT S.level, ROUND(AVG(S.age), 2) AS avg_age
FROM Student S
GROUP BY S.level;
```

8)
```
SELECT S.level, ROUND(AVG(S.age), 2) AS avg_age
FROM Student S
WHERE S.level <> 'JR'
GROUP BY S.level;
```

9)
```
SELECT F.fname, COUNT(C.name) AS total_classes
FROM Faculty F
JOIN Class C ON F.fid = C.fid
WHERE C.room = 'R128'
GROUP BY F.fid, F.fname
HAVING COUNT(DISTINCT C.room) = 1;
```

10)
```
WITH EnrollCount AS (
   SELECT S.snum, S.sname, COUNT(E.cname) AS num_classes
   FROM Student S
   JOIN Enrolled E ON S.snum = E.snum
   GROUP BY S.snum, S.sname
)
SELECT sname
FROM EnrollCount
WHERE num_classes = (SELECT MAX(num_classes) FROM EnrollCount);
```

11)
```
SELECT S.sname
FROM Student S
WHERE S.snum NOT IN (
   SELECT E.snum
```

```
    FROM Enrolled E
);
12) WITH LevelCount AS (
    SELECT S.age, S.level, COUNT(*) AS level_count
    FROM Student S
    GROUP BY S.age, S.level
),
MaxLevel AS (
    SELECT L.age, MAX(L.level_count) AS max_count
    FROM LevelCount L
    GROUP BY L.age
)
SELECT L.age, L.level
FROM LevelCount L
JOIN MaxLevel M ON L.age = M.age AND L.level_count = M.max_count;
```

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.

Ans:

```
1) SELECT DISTINCT P.pname
FROM Parts P
JOIN Catalog C ON P.pid = C.pid;
2) SELECT S.sname
FROM Suppliers S
```

```
WHERE NOT EXISTS (
   SELECT P.pid
   FROM Parts P
   WHERE P.pid NOT IN (
      SELECT C.pid
      FROM Catalog C
      WHERE C.sid = S.sid
   )
);
```

3)
```
SELECT S.sname
FROM Suppliers S
WHERE NOT EXISTS (
   SELECT P.pid
   FROM Parts P
   WHERE P.color = 'red'
     AND P.pid NOT IN (
        SELECT C.pid
        FROM Catalog C
        WHERE C.sid = S.sid
     )
);
```

4)
```
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 P.pid NOT IN (
     SELECT C2.pid
     FROM Catalog C2
     WHERE C2.sid != C1.sid
  );
```

```
5) SELECT DISTINCT C.sid
FROM Catalog C
JOIN (
   SELECT pid, AVG(cost) AS avg_cost
   FROM Catalog
   GROUP BY pid
) AvgCosts ON C.pid = AvgCosts.pid
WHERE C.cost > AvgCosts.avg_cost;
6) 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) SELECT S.sid
FROM Suppliers S
WHERE NOT EXISTS (
   SELECT P.pid
   FROM Parts P
   WHERE P.color != 'red'
     AND P.pid IN (
        SELECT C.pid
        FROM Catalog C
        WHERE C.sid = S.sid
     )
);
8) SELECT DISTINCT C1.sid
FROM Catalog C1
```

```
JOIN Parts P1 ON C1.pid = P1.pid
WHERE P1.color = 'red'
  AND EXISTS (
     SELECT 1
     FROM Catalog C2
     JOIN Parts P2 ON C2.pid = P2.pid
     WHERE P2.color = 'green' AND C2.sid = C1.sid
  );
9) SELECT DISTINCT C.sid
FROM Catalog C
JOIN Parts P ON C.pid = P.pid
WHERE P.color = 'red' OR P.color = 'green';
10) SELECT S.sname, COUNT(C.pid) AS total_parts
FROM Suppliers S
JOIN Catalog C ON S.sid = C.sid
JOIN Parts P ON C.pid = P.pid
WHERE NOT EXISTS (
   SELECT 1
   FROM Catalog C1
   JOIN Parts P1 ON C1.pid = P1.pid
   WHERE C1.sid = S.sid AND P1.color != 'green'
)
GROUP BY S.sname;
11) SELECT S.sname, MAX(C.cost) AS max_cost
FROM Suppliers S
JOIN Catalog C ON S.sid = C.sid
WHERE S.sid IN (
   SELECT C1.sid
   FROM Catalog C1
   JOIN Parts P1 ON C1.pid = P1.pid
   WHERE P1.color = 'red'
```

```
)
AND S.sid IN (
   SELECT C2.sid
   FROM Catalog C2
   JOIN Parts P2 ON C2.pid = P2.pid
   WHERE P2.color = 'green'
)
GROUP BY S.sname;
```

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.

Ans:

```
1) 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) SELECT C.eid, MAX(A.cruisingrange) AS max_cruisingrange
FROM Certified C
JOIN Aircraft A ON C.aid = A.aid
GROUP BY C.eid
HAVING COUNT(C.aid) > 3;
3) 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) SELECT A.aname, AVG(E.salary) AS avg_salary
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) 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) SELECT A.aid
FROM Aircraft A
WHERE A.cruisingrange >= (
    SELECT F.distance
    FROM Flights F
```

```
   WHERE F.from = 'Los Angeles' AND F.to = 'Chicago'
);
7) SELECT F.flno
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
       WHERE C.eid = E.eid AND C.aid IN (
          SELECT A.aid
          FROM Aircraft A
          WHERE A.cruisingrange >= F.distance
       )
    )
);
8) 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) SELECT F1.departs
FROM Flights F1, Flights F2, Flights F3
```

```
WHERE F1.from = 'Madison'
  AND (
     (F1.to = 'New York' AND F1.arrives <= '18:00:00')
     OR (
        F1.to = F2.from AND F2.to = 'New York' AND F2.arrives <= '18:00:00'
     )
     OR (
        F1.to = F2.from AND F2.to = F3.from AND F3.to = 'New York'
        AND F3.arrives <= '18:00:00'
     )
  );
10) SELECT
    (SELECT AVG(E1.salary)
     FROM Employees E1
     JOIN Certified C ON E1.eid = C.eid) -
    (SELECT AVG(E2.salary)
     FROM Employees E2) AS salary_difference;
11) SELECT E.ename, E.salary
FROM Employees E
WHERE E.eid NOT IN (SELECT C.eid FROM Certified C)
  AND E.salary > (
     SELECT AVG(E1.salary)
     FROM Employees E1
     JOIN Certified C ON E1.eid = C.eid
  );
12) 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) SELECT E.ename
FROM Employees E
JOIN Certified C ON E.eid = C.eid
JOIN Aircraft A ON C.aid = A.aid
WHERE NOT EXISTS (
   SELECT C1.aid
   FROM Certified C1
   JOIN Aircraft A1 ON C1.aid = A1.aid
   WHERE C1.eid = E.eid AND A1.cruisingrange <= 1000
)
GROUP BY E.eid, E.ename
HAVING COUNT(DISTINCT A.aid) >= 2;
14) SELECT DISTINCT E.ename
FROM Employees E
WHERE NOT EXISTS (
   SELECT C1.aid
   FROM Certified C1
   JOIN Aircraft A1 ON C1.aid = A1.aid
   WHERE C1.eid = E.eid AND A1.cruisingrange <= 1000
)
AND EXISTS (
   SELECT C2.aid
   FROM Certified C2
   JOIN Aircraft A2 ON C2.aid = A2.aid
   WHERE C2.eid = E.eid AND A2.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.

Ans:

```
1) CREATE TABLE Emp (
   eid INTEGER PRIMARY KEY,
   ename VARCHAR(100),
   age INTEGER,
   salary REAL,
   CONSTRAINT salary_check CHECK (salary >= 10000)
);
2) CREATE TABLE Dept (
   did INTEGER PRIMARY KEY,
   budget REAL,
   managerid INTEGER,
   CONSTRAINT manager_age_check
      CHECK (managerid IN (SELECT eid FROM Emp WHERE age > 30))
);
3) CREATE ASSERTION manager_age_assertion
   CHECK (NOT EXISTS (
      SELECT 1
      FROM Dept D
      JOIN Emp E ON D.managerid = E.eid
      WHERE E.age <= 30
   ));
4) -- First, find employees whose salary exceeds the manager's salary for any department
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
):
```

16)

1. ALTER TABLE Emp
   ADD CONSTRAINT min_salary
   CHECK (salary >= 1000);

1. ALTER TABLE Dept
   ADD CONSTRAINT mgr_is_emp
   FOREIGN KEY (managerid) REFERENCES Emp(eid);

2. CREATE TRIGGER check_total_pct_time
   AFTER INSERT OR UPDATE OF pct_time ON Works
   FOR EACH ROW
   BEGIN
    DECLARE total_pct INTEGER;
    SELECT SUM(pct_time) INTO total_pct
    FROM Works
    WHERE eid = :NEW.eid;
    IF total_pct > 100 THEN
     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Total percentage exceeds 100%';
    END IF;
     END;

3. CREATE TRIGGER check_manager_salary
   AFTER INSERT OR UPDATE OF salary ON Emp
   FOR EACH ROW
   BEGIN
    DECLARE emp_salary INTEGER;
    SELECT salary INTO emp_salary
    FROM Emp
    WHERE eid = :NEW.eid;

    DECLARE mgr_salary INTEGER;
    SELECT salary INTO mgr_salary
    FROM Dept
    WHERE managerid = :NEW.eid;

    IF emp_salary >= mgr_salary THEN
     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Manager salary must be higher';
    END IF;
   END;

4. CREATE TRIGGER update_manager_salary_on_raise
   AFTER UPDATE OF salary ON Emp
   FOR EACH ROW
   BEGIN

```
 DECLARE mgr_id INTEGER;
 SELECT managerid INTO mgr_id
 FROM Dept
 WHERE managerid = :OLD.eid;
  UPDATE Emp
 SET salary = GREATEST(salary, :NEW.salary)
 WHERE eid = mgr_id;
END;
```

5.
```
CREATE TRIGGER update_manager_salary_and_budget_on_raise
AFTER UPDATE OF salary ON Emp
FOR EACH ROW
BEGIN
 DECLARE mgr_id INTEGER;
 SELECT managerid INTO mgr_id
 FROM Dept
 WHERE managerid = :OLD.eid;

 UPDATE Emp
 SET salary = GREATEST(salary, :NEW.salary)
 WHERE eid = mgr_id;

 DECLARE dept_id INTEGER;
 SELECT did INTO dept_id
 FROM Works
 WHERE eid = :OLD.eid;

 DECLARE total_salary INTEGER;
 SELECT SUM(salary) INTO total_salary
 FROM Emp
 JOIN Works ON Emp.eid = Works.eid
 WHERE Works.did = dept_id;

 UPDATE Dept
 SET budget = GREATEST(budget, total_salary)
 WHERE did = dept_id;
 END;
```

17)

Scenario 1: Set Membership (IN, NOT IN)
Find customers who have placed orders after '2023-12-31':

```
SELECT customer_name
FROM Customers
WHERE customer_id IN (
   SELECT customer_id
```

```
    FROM Orders
    WHERE order_date > '2023-12-31'
);
```

Find customers who have not placed any orders:

```
SELECT customer_name
FROM Customers
WHERE customer_id NOT IN (
    SELECT customer_id
    FROM Orders
);
```

Scenario 2: Set Comparison

- **Find customers who have placed orders with a higher average amount than the average order amount of all customers:**

```
SELECT customer_name
FROM Customers
WHERE customer_id IN (
    SELECT customer_id
    FROM Orders
    GROUP BY customer_id
    HAVING AVG(amount) > (
        SELECT AVG(amount)
        FROM Orders
    )
);
```

Scenario 3: Set Cardinality

- **Find customers who have placed exactly 3 orders:**

```
SELECT customer_name
FROM Customers
```

```
WHERE customer_id IN (
  SELECT customer_id
  FROM Orders
  GROUP BY customer_id
  HAVING COUNT(*) = 3
);
```

18) 1. Simple View:

```
CREATE VIEW CustomerOrders AS
SELECT c.customer_name, o.order_id, o.order_date
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id;
```

2. View with a WHERE Clause:
   ```
   CREATE VIEW HighValueOrders AS
   SELECT *
   FROM Orders
   WHERE amount > 1000;
   ```

3. View from Multiple Tables:
   ```
   CREATE VIEW CustomerCityOrders AS
   SELECT c.customer_name, c.city, o.order_id, o.order_date
   FROM Customers c
   JOIN Orders o ON c.customer_id = o.customer_id;
   ```

19) **Strengths of Triggers:**

- **Complex Constraints:** Can enforce intricate rules beyond basic data validation.
- **Procedural Logic:** Allows execution of custom code, enabling actions like logging, auditing, and notifications.
- **Timing Control:** Can be triggered before or after specific events, providing precise control over data modification.
- **Cascading Actions:** Can initiate chain reactions to maintain data consistency across multiple tables.

Weaknesses of Triggers:

- **Complexity:** Can be difficult to write, test, and maintain, especially for complex scenarios.
- **Performance Impact:** Can degrade database performance if not optimized.

- **Error Handling:** Errors within triggers can lead to data inconsistencies.
- **Debugging Challenges:** Can be difficult to debug due to their event-driven nature.

20)

1. CREATE OR REPLACE FUNCTION calculate_salary(p_basic_salary NUMBER)

RETURN NUMBER

IS

 v_da NUMBER := p_basic_salary * 0.1;

 v_hra NUMBER := p_basic_salary * 0.2;

 v_total_salary NUMBER;

BEGIN

 v_total_salary := p_basic_salary + v_da + v_hra;

 RETURN v_total_salary;

END;

/

2. CREATE OR REPLACE TRIGGER trg_after_insert_employee

AFTER INSERT ON employees

FOR EACH ROW

DECLARE

 v_message VARCHAR2(100);

BEGIN

 v_message := 'New employee inserted: ' || :NEW.employee_name;

 DBMS_OUTPUT.PUT_LINE(v_message);

END;

/

3. CREATE OR REPLACE TRIGGER trg_before_delete_employee

BEFORE DELETE ON employees

FOR EACH ROW

```
DECLARE
 v_message VARCHAR2(100);
BEGIN
 v_message := 'Deleting employee: ' || :OLD.employee_name;
 DBMS_OUTPUT.PUT_LINE(v_message);
END;
/
```

4.
```
CREATE OR REPLACE TRIGGER trg_check_salary
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
DECLARE
 v_message VARCHAR2(100);
BEGIN
 IF :NEW.salary <= 0 THEN
  RAISE_APPLICATION_ERROR(-20001, 'Salary cannot be zero or negative');
 END IF;
END;
/
```

5.
```
CREATE OR REPLACE TRIGGER trg_check_city
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
DECLARE
 v_message VARCHAR2(100);
BEGIN
 IF :NEW.city <> 'Pune' THEN
  RAISE_APPLICATION_ERROR(-20002, 'Employee must reside in Pune');
 END IF;
END;
/
```

21)

1.
```
UPDATE Deposit
SET Amount = Amount * 1.2;
```

2. UPDATE Deposit

SET Amount = Amount * 1.1

WHERE BName = 'Manjari';

3. UPDATE Deposit d

SET d.Amount = d.Amount * 1.1

WHERE EXISTS (

  SELECT 1

  FROM Customers c

  WHERE c.CName = d.CName

  AND c.City = 'Pune'

  AND d.BName = 'Bombay'

);

4. UPDATE Emp
SET Deptno = (
  SELECT Deptno
  FROM Emp
  WHERE Empno = 7844
)
WHERE Job = (
  SELECT Job
  FROM Emp
  WHERE Empno = 7788
);

5. UPDATE Emp
SET Deptno = 30; -- Replace 30 with the desired department number

6. UPDATE Deposit a
SET Amount = Amount - 10
WHERE CName = 'Anil'
AND EXISTS (
  SELECT 1
  FROM Deposit s
  WHERE s.CName = 'Sunil'
  AND s.BName = a.BName
);

UPDATE Deposit s

```
SET Amount = Amount + 10
WHERE CName = 'Sunil'
AND EXISTS (
   SELECT 1
   FROM Deposit a
   WHERE a.CName = 'Anil'
   AND a.BName = s.BName
);
```

7.
```
UPDATE Deposit d
SET Amount = Amount + 100
WHERE Amount = (
   SELECT MAX(Amount)
   FROM Deposit
   WHERE BName = d.BName
);
```

8.
```
DELETE FROM Deposit
WHERE CName = 'Vijay';
```

9.
```
DELETE FROM Borrow b
WHERE EXISTS (
   SELECT 1
   FROM (
      SELECT BName, AVG(Amount) AS AvgLoan
      FROM Borrow
      GROUP BY BName
      HAVING AVG(Amount) < 1000
   ) a
   WHERE a.BName = b.BName
);
```

22) Scenario 1: Set Membership (IN, NOT IN)
```
    SELECT customer_name
FROM Customers
WHERE customer_id IN (
   SELECT customer_id
   FROM Orders
   WHERE order_date > '2023-12-31'
);
```

Scenario 2: Set Comparison
```
SELECT customer_name
FROM Customers c
WHERE (
   SELECT AVG(amount)
   FROM Orders o
   WHERE o.customer_id = c.customer_id
```

```
    ) > (
      SELECT AVG(amount)
      FROM Orders
    );
```

Scenario 3: Set Cardinality

```
SELECT customer_name
FROM Customers c
WHERE 3 = (
  SELECT COUNT(*)
  FROM Orders o
  WHERE o.customer_id = c.customer_id
);
```

23) 1.

```
SELECT 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');
```

2.

```
SELECT s.sname
FROM Sailors s
WHERE EXISTS (
  SELECT 1
  FROM Reserves r1
  JOIN Boats b1 ON r1.bid = b1.bid
  WHERE r1.sid = s.sid
  AND b1.color = 'red'
)
AND EXISTS (
  SELECT 1
  FROM Reserves r2
  JOIN Boats b2 ON r2.bid = b2.bid
```

```
   WHERE r2.sid = s.sid
   AND b2.color = 'green'
);
```

3.
```
SELECT s.sname
FROM Sailors s
GROUP BY s.sid
HAVING COUNT(DISTINCT r.bid) >= 2;
```

4.
```
SELECT s.sid
FROM Sailors s
WHERE s.age > 20
AND NOT EXISTS (
   SELECT 1
   FROM Reserves r
   JOIN Boats b ON r.bid = b.bid
   WHERE r.sid = s.sid
   AND b.color = 'red'
);
```

5.
```
SELECT s.sname
FROM Sailors s
WHERE NOT EXISTS (
   SELECT b.bid
   FROM Boats b
   WHERE NOT EXISTS (
      SELECT 1
      FROM Reserves r
      WHERE r.sid = s.sid
      AND r.bid = b.bid
   )
);
```

6.
```
SELECT s.sname
FROM Sailors s
WHERE NOT EXISTS (
```

```
    SELECT b.bid
    FROM Boats b
    WHERE b.bname = 'Interlake'
    AND NOT EXISTS (
        SELECT 1
        FROM Reserves r
        WHERE r.sid = s.sid
        AND r.bid = b.bid
    )
);
```

7.
```
SELECT *
FROM Sailors
WHERE rating > 7;
```

8.
```
SELECT s.sname, r.bid, r.day
FROM Sailors s
JOIN Reserves r ON s.sid = r.sid;
```

9.
```
SELECT s.sname
FROM Sailors s
WHERE NOT EXISTS (
    SELECT b.bid
    FROM Boats b
    WHERE b.color = 'red'
    AND NOT EXISTS (
        SELECT 1
        FROM Reserves r
        WHERE r.sid = s.sid
        AND r.bid = b.bid
    )
);
```

10. SELECT sname, age

FROM Sailors;

24) 1. SELECT

(SELECT AVG(salary) FROM Employees WHERE eid IN (SELECT eid FROM Certified)) -

AVG(salary)

FROM Employees;

2. SELECT ename, salary

FROM Employees e

WHERE salary > (

SELECT AVG(salary)

FROM Employees p

WHERE p.eid IN (SELECT eid FROM Certified)

)

AND e.eid NOT IN (SELECT eid FROM Certified);

3. SELECT e.ename

FROM Employees e

WHERE NOT EXISTS (

SELECT *

FROM Certified c

JOIN Aircraft a ON c.aid = a.aid

WHERE c.eid = e.eid

AND a.cruisingrange <= 1000

);

4. SELECT e.ename

FROM Employees e

WHERE (

SELECT COUNT(*)

```
    FROM Certified c
    JOIN Aircraft a ON c.aid = a.aid
    WHERE c.eid = e.eid
    AND a.cruisingrange > 1000
) >= 2
AND NOT EXISTS (
    SELECT *
    FROM Certified c
    JOIN Aircraft a ON c.aid = a.aid
    WHERE c.eid = e.eid
    AND a.cruisingrange <= 1000
);
```

5.
```
SELECT e.ename
FROM Employees e
WHERE NOT EXISTS (
    SELECT *
    FROM Certified c
    JOIN Aircraft a ON c.aid = a.aid
    WHERE c.eid = e.eid
    AND a.cruisingrange <= 1000
)
AND EXISTS (
    SELECT *
    FROM Certified c
    JOIN Aircraft a ON c.aid = a.aid
    WHERE c.eid = e.eid
    AND a.aname LIKE '%Boeing%'
);
```

25) 1. 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';

2. SELECT MAX(s.age)

FROM Student s

WHERE s.major = 'History'

OR s.snum IN (

SELECT e.snum

FROM Enrolled e

JOIN Class c ON e.cname = c.name

JOIN Faculty f ON c.fid = f.fid

WHERE f.fname = 'I. Teacher'

);

3. SELECT c.name

FROM Class c

WHERE c.room = 'R128'

OR (

SELECT COUNT(*)

FROM Enrolled e

WHERE e.cname = c.name

) >= 5;

4. SELECT s.sname

FROM Student s

JOIN Enrolled e1 ON s.snum = e1.snum

```
JOIN Class c1 ON e1.cname = c1.name
JOIN Enrolled e2 ON s.snum = e2.snum
JOIN Class c2 ON e2.cname = c2.name
WHERE c1.meetsat = c2.meetsat
  AND c1.name <> c2.name;
```

```
5. SELECT f.fname
FROM Faculty f
WHERE NOT EXISTS (
    SELECT c.room
    FROM Class c
    WHERE NOT EXISTS (
        SELECT 1
        FROM Class c2
        JOIN Faculty f2 ON c2.fid = f2.fid
        WHERE c2.room = c.room
          AND f2.fname = f.fname
    )
);
```

```
6. SELECT f.fname
FROM Faculty f
WHERE (
    SELECT SUM(
        SELECT COUNT(*)
        FROM Enrolled e
        WHERE e.cname = c.name
    )
    FROM Class c
    WHERE c.fid = f.fid
) < 5;
```

7. SELECT s.level, AVG(s.age)
FROM Student s
GROUP BY s.level;

8. SELECT s.level, AVG(s.age)
FROM Student s
WHERE s.level <> 'JR'
GROUP BY s.level;

9. SELECT s.sname
FROM Student s
WHERE (
  SELECT COUNT(*)
  FROM Enrolled e
  WHERE e.snum = s.snum
) = (
  SELECT MAX(cnt)
  FROM (
    SELECT s2.snum, COUNT(*) AS cnt
    FROM Student s2
    JOIN Enrolled e2 ON s2.snum = e2.snum
    GROUP BY s2.snum
) AS MaxEnrollments
);

10. SELECT s.sname
FROM Student s
WHERE NOT EXISTS (
  SELECT 1
  FROM Enrolled e
  WHERE e.snum = s.snum

```
);
```

26) 1. INSERT INTO Customer (customer_id, name, email, address)
VALUES (nextval('customer_seq'), 'New Customer', 'new_customer@example.com', '123 Main St');

```
ALTER TABLE Customer
ADD CONSTRAINT unique_email
UNIQUE (email);
2. CREATE TABLE Employee (
   employee_name VARCHAR(50) PRIMARY KEY,
   street VARCHAR(100),
   city VARCHAR(50)
);

CREATE TABLE Company (
   company_name VARCHAR(50) PRIMARY KEY,
   city VARCHAR(50)
);

CREATE TABLE Works (
   employee_name VARCHAR(50),
   company_name VARCHAR(50),
   salary DECIMAL(10,2),
   PRIMARY KEY (employee_name, company_name),
   FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
   FOREIGN KEY (company_name) REFERENCES Company(company_name)
);

CREATE TABLE Manages (
   employee_name VARCHAR(50),
   manager_name VARCHAR(50),
```

```
    PRIMARY KEY (employee_name, manager_name),
    FOREIGN KEY (employee_name) REFERENCES Employee(employee_name),
    FOREIGN KEY (manager_name) REFERENCES Employee(employee_name)
);
```

27) a.
```
SELECT c.CustName
FROM Customer c
JOIN HLoan h ON c.Custid = h.Custid
JOIN VLoan v ON c.Custid = v.Custid;
```

b.
```
SELECT c.Custid, SUM(v.Amount) AS TotalVLoan
FROM Customer c
JOIN VLoan v ON c.Custid = v.Custid
GROUP BY c.Custid;
```

c.
```
CREATE TRIGGER trg_HLoan_Insert
AFTER INSERT ON HLoan
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('New HLoan inserted:');
  DBMS_OUTPUT.PUT_LINE('HLoan ID: ' || :NEW.HLoanid);
  DBMS_OUTPUT.PUT_LINE('Amount: ' || :NEW.Amount);
  DBMS_OUTPUT.PUT_LINE('Customer ID: ' || :NEW.Custid);
END;
/

CREATE TRIGGER trg_VLoan_Insert
AFTER INSERT ON VLoan
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('New VLoan inserted:');
```

```
  DBMS_OUTPUT.PUT_LINE('VLoan ID: ' || :NEW.VLoanid);
  DBMS_OUTPUT.PUT_LINE('Amount: ' || :NEW.Amount);
  DBMS_OUTPUT.PUT_LINE('Customer ID: ' || :NEW.Custid);
END;
/
```

d.
```
CREATE VIEW CustomerTotalLoan AS
SELECT
    c.Custid,
    c.CustName,
    (h.Amount + v.Amount) AS TotalLoanAmount
FROM Customer c
JOIN HLoan h ON c.Custid = h.Custid
JOIN VLoan v ON c.Custid = v.Custid;
```

28) a.
```
SELECT c.CustName
FROM Customer c
JOIN Loan l ON c.Custid = l.Custid
JOIN Account a ON c.Custid = a.Custid;
```

b.
```
CREATE OR REPLACE FUNCTION getBal(p_Custid INT)
RETURN DECIMAL(10,2)
IS
  v_total_balance DECIMAL(10,2) := 0;
BEGIN
  SELECT SUM(Accbal) INTO v_total_balance
  FROM Account
  WHERE Custid = p_Custid;
  RETURN v_total_balance;
END;
/
```

c. CREATE OR REPLACE PROCEDURE print_high_balance_customers

IS

CURSOR c_customers IS

SELECT CustName, getBal(Custid) AS TotalBalance

FROM Customer

HAVING getBal(Custid) > 10000;

v_name Customer.CustName%TYPE;

v_balance DECIMAL(10,2);

BEGIN

OPEN c_customers;

LOOP

FETCH c_customers INTO v_name, v_balance;

EXIT WHEN c_customers%NOTFOUND;

DBMS_OUTPUT.PUT_LINE('Customer Name: ' || v_name || ', Total Balance: ' || v_balance);

END LOOP;

CLOSE c_customers;

END;

/

To execute the procedure:

EXECUTE print_high_balance_customers;

29) 1. import sqlite3

conn = sqlite3.connect('university.db')

cursor = conn.cursor()

2. cursor.execute('''

CREATE TABLE IF NOT EXISTS Students (

StudentID INTEGER PRIMARY KEY AUTOINCREMENT,

StudentName TEXT,

```
    Age INTEGER,
    MajorID INTEGER
  )
''')
conn.commit()
```

3. CRUD Operations

CREATE:

```
def insert_student(student_name, age, major_id):
  cursor.execute("INSERT INTO Students (StudentName, Age, MajorID) VALUES (?, ?, ?)",
          (student_name, age, major_id))
  conn.commit()
```

READ:

```
def get_all_students():
  cursor.execute("SELECT * FROM Students")
  rows = cursor.fetchall()
  return rows

def get_student_by_id(student_id):
  cursor.execute("SELECT * FROM Students WHERE StudentID = ?", (student_id,))
  row = cursor.fetchone()
  return row
```

UPDATE:

```
def update_student(student_id, new_name, new_age, new_major_id):
  cursor.execute("UPDATE Students SET StudentName = ?, Age = ?, MajorID = ? WHERE StudentID =
?",
          (new_name, new_age, new_major_id, student_id))
  conn.commit()
```

DELETE:

```
def delete_student(student_id):
  cursor.execute("DELETE FROM Students WHERE StudentID = ?", (student_id,))
```

```
    conn.commit()
```

CLOSING THE CONNECTION :

```
conn.close()
```

30)

```
import sqlite3

def insert_students_with_savepoint():
    conn = sqlite3.connect('university.db')
    cursor = conn.cursor()

    try:
        # Start a transaction
        conn.execute("BEGIN TRANSACTION")

        # Insert the first student
        cursor.execute("INSERT INTO Students (StudentName, Age, MajorID) VALUES (?, ?, ?)", ('Alice',
20, 1))
        conn.commit()

        # Set a savepoint
        conn.execute("SAVEPOINT first_student")

        # Insert the second student
        cursor.execute("INSERT INTO Students (StudentName, Age, MajorID) VALUES (?, ?, ?)", ('Bob', 21,
2))

        # Commit the transaction
        conn.commit()

    except Exception as e:
        # Rollback to the savepoint if an error occurs
```

```
        conn.rollback()
        print("Error occurred:", e)

    finally:
        conn.close()

if __name__ == "__main__":
    insert_students_with_savepoint()
```