Subject: DBMSL
Assignment List for practice

SQL

Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym.

Create following tables with primary and foreign key and solve the queries given below

Person (driver_id, name, address)
Car (license, model, year)
Accident (report_no, date_acc, location)
Owns (driver_id, license)
Participated (driver_id, model, report_no, damage_amount)

Employee (employee_name,street,city)
Works (employee_name, company_name,salary)
Company (company_name, city)
Manages (employee_name, manager_name)

```
CREATE TABLE Person (
   driver_id INT PRIMARY KEY,
   name VARCHAR(50),
   address VARCHAR(100)
);

CREATE TABLE Car (
   license VARCHAR(20) PRIMARY KEY,
   model VARCHAR(50),
   year INT
);

CREATE TABLE Accident (
   report_no INT PRIMARY KEY,
   date_acc DATE,
   location VARCHAR(100)
);

CREATE TABLE Owns (
   driver_id INT,
   license VARCHAR(20),
   PRIMARY KEY (driver_id, license),
   FOREIGN KEY (driver_id) REFERENCES Person(driver_id),
```

```
    FOREIGN KEY (license) REFERENCES Car(license)
);

CREATE TABLE Participated (
    driver_id INT,
    model VARCHAR(50),
    report_no INT,
    damage_amount DECIMAL(10,2),
    PRIMARY KEY (driver_id, model, report_no),
    FOREIGN KEY (driver_id) REFERENCES Person(driver_id),
    FOREIGN KEY (model) REFERENCES Car(model),
    FOREIGN KEY (report_no) REFERENCES Accident(report_no)
);

CREATE TABLE Employee (
    employee_name VARCHAR(50) PRIMARY KEY,
    street VARCHAR(100),
    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 Company (
    company_name VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50)
);

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)
);
```

1) Create view with the employee_name, company_name by using above tables.
   Create view emp (employee_name, company_name) AS Select e.employee_name, c.company_name From Employee e, Company c Where e.city=c.city
2) Create index for employee & participated table. Create INDEX emp on Employee (employee_name,street,city);
   Create INDEX part on Participated (driver_id, model, report_no, damage_amount);
3) Create sequence for person & insert 4 records using sequence. CREATE SEQUENCE driver_id_seq START WITH 1 INCREMENT BY 1 NOMAXVALUE NOCACHE;
   INSERT INTO Person (name, address) VALUES ('Alice', '123 Main St'), ('Bob', '456 Elm St'), ('Charlie', '789 Oak St'), ('David', '101 Pine St');
4) Create the synonym for table participated & company. Display the record using this table. Update the record using the synonym tables. CREATE SYNONYM accident_details FOR participated; CREATE SYNONYM company_info FOR company;
   SELECT * FROM accident_details; SELECT * FROM company_info;
   UPDATE accident_details SET damage_amount = 1000 WHERE driver_id = 123; UPDATE company_info SET city = 'New York' WHERE company_name = 'TechCorp';
5) Drop index from employee table. Drop INDEX emp

Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym.

employee (person name, street, city)
works (person name, company name, salary)
company (company name, city)
branch (branch name, branch city, assets)
customer (customer name, customer street, customer city)
loan (loan number, branch name, amount)
borrower (customer name, loan number)
account (account number, branch name, balance)
depositor (customer name, account number)

```
CREATE TABLE Employee (
    person_name VARCHAR(50) PRIMARY KEY,
    street VARCHAR(100),
    city VARCHAR(50)
);

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

```
);

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

CREATE TABLE Branch (
    branch_name VARCHAR(50),
    branch_city VARCHAR(50),
    assets DECIMAL(15,2),
    PRIMARY KEY (branch_name)
);

CREATE TABLE Customer (
    customer_name VARCHAR(50) PRIMARY KEY,
    customer_street VARCHAR(100),
    customer_city VARCHAR(50)
);

CREATE TABLE Loan (
    loan_number INT PRIMARY KEY,
    branch_name VARCHAR(50),
    amount DECIMAL(10,2),
    FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

CREATE TABLE Borrower (
    customer_name VARCHAR(50),
    loan_number INT,
    PRIMARY KEY (customer_name, loan_number),
    FOREIGN KEY (customer_name) REFERENCES Customer(customer_name),
    FOREIGN KEY (loan_number) REFERENCES Loan(loan_number)
);

CREATE TABLE Account (
    account_number INT PRIMARY KEY,
    branch_name VARCHAR(50),
    balance DECIMAL(10,2),
    FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

CREATE TABLE Depositor (
```

```
    customer_name VARCHAR(50),
    account_number INT,
    PRIMARY KEY (customer_name, account_number),
    FOREIGN KEY (customer_name) REFERENCES Customer(customer_name),
    FOREIGN KEY (account_number) REFERENCES Account(account_number)
);
```

1) Create view with the Customer name, customer street by using above tables. CREATE VIEW CustomerInfo AS SELECT customer_name, customer_street FROM Customer;
2) Create index for loan number. Create INDEX lno on loan(loan number);
3) Create sequence for account & insert 4 records using sequence. CREATE SEQUENCE account_seq START WITH 1000 INCREMENT BY 1 NOMAXVALUE NOCACHE; INSERT INTO Account (account_number, branch_name, balance)
   VALUES
   (nextval('account_seq'), 'BranchA', 1000),
   (nextval('account_seq'), 'BranchB', 2000),
   (nextval('account_seq'), 'BranchA', 3000),
   (nextval('account_seq'), 'BranchB', 4000);
4) Create the synonym for table depositor & borrower. Display the record using this table. Update the record using the synonym tables.CREATE SYNONYM account_holder FOR Depositor; CREATE SYNONYM loan_taker FOR Borrower; SELECT * FROM account_holder; SELECT * FROM loan_taker; UPDATE account_holder SET account_number = 1001 WHERE customer_name = 'Alice'; UPDATE loan_taker SET loan_number = 2001 WHERE customer_name = 'Bob';
5) Drop synonym for depositor table. DROP SYNONYM account_holder;

Design at least 10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, delete with operators, functions, and set operator.

1) Create table Department with fields deptno, dname, location. CREATE TABLE Department (Deptno INT PRIMARY KEY, Dname VARCHAR(20), Location VARCHAR(20));
2) Insert records in the department table.

Deptno	Dname	Location
10	Accounting	Mumbai
20	Research	Pune
30	Sales	Nashik
40	Operations	Nagpur

INSERT INTO Department (Deptno, Dname, Location)
VALUES
(10, 'Accounting', 'Mumbai'),
(20, 'Research', 'Pune'),
(30, 'Sales', 'Nashik'),

(40, 'Operations', 'Nagpur');

3) List the department information. Select * from department;
4) Create table employee with Empno, Ename, Job, Mgr, Joined_date, Salary, Commission, Deptno CREATE TABLE Employee (Empno INT PRIMARY KEY,Ename VARCHAR(20),Job VARCHAR(20), Mgr INT, Joined_date DATE, Salary INT, Commission INT, Deptno INT, Address VARCHAR(20));
5) Insert records into employee table.

Empno	Ename	Job	Mgr	Joined_date	Salary	Commission	Deptno	Address
1001	Nilesh joshi	Clerk	1005	17-dec-95	2800	600	20	Nashik
1002	Avinashpawar	Salesman	1003	20-feb-96	5000	1200	30	Nagpur
1003	Amit kumar	Manager	1004	2-apr-86	2000	----	30	Pune
1004	Nitinkulkarni	President	--	19-apr-86	50000	----	10	Mumbai
1005	Niraj Sharma	Analyst	1003	3-dec-98	12000	----	20	Satara
1006	Pushkardeshpande	Salesman	1003	1-sep-96	6500	1500	30	Pune
1007	Sumitpatil	Manager	1004	1-may-91	25000	----	20	Mumbai
1008	Ravi sawant	Analyst	1007	17-nov-95	10000	----	---	Amaravati

INSERT INTO Employee (Empno, Ename, Job, Mgr, Joined_date, Salary, Commission, Deptno, Address)
VALUES
(1001, 'Nilesh Joshi', 'Clerk', 1005, '1995-12-17', 2800, 600, 20, 'Nashik'),
(1002, 'Avinash Pawar', 'Salesman', 1003, '1996-02-20', 5000, 1200, 30, 'Nagpur'),
(1003, 'Amit Kumar', 'Manager', 1004, '1986-04-02', 2000, NULL, 30, 'Pune'),
(1004, 'Nitin Kulkarni', 'President', NULL, '1986-04-19', 50000, NULL, 10, 'Mumbai'),
(1005, 'Niraj Sharma', 'Analyst', 1003, '1998-12-03', 12000, NULL, 20, 'Satara'),
(1006, 'Pushkar Deshpande', 'Salesman', 1003, '1996-09-01', 6500, 1500, 30, 'Pune'),
(1007, 'Sumit Patil', 'Manager', 1004, '1991-05-01', 25000, NULL, 20, 'Mumbai'),
(1008, 'Ravi Sawant', 'Analyst', 1007, '1995-11-17', 10000, NULL,
20, 'Amaravati');

6) Write a query to display employee information. Write a name of column explicitly.
SELECT Empno, Ename, Job, Mgr, Joined_date, Salary, Commission, Deptno, Address
FROM Employee;

7) Add column phone number in the employee table and add not null constraints. ALTER TABLE Employee ADD Phone_number VARCHAR(20) NOT NULL;
8) Create a query to display unique jobs from the table. SELECT DISTINCT Job FROM Employee;
9) Change the location of dept 40 to Banglore instead of Nagpur.UPDATE Department SET Location = 'Bangalore' WHERE Deptno = 40;
10) Delete Pushkar deshpande from employee table.DELETE FROM Employee WHERE Ename = 'Pushkar Deshpande';
11) Create a table department_temp table from deptarment table, only create the structure not content.CREATE TABLE Department_temp (Deptno INT PRIMARY KEY, Dname VARCHAR(20),Location VARCHAR(20));
12) Insert rows into department_temp table from deptarment table INSERT INTO Department_temp SELECT * FROM Department;
13) Change the name of the employees 1003 to Nikhil Gosavi.UPDATE Employee SET Ename = 'Nikhil Gosavi' WHERE Empno = 1003;
14) Display the list of employee whose salary between 5000 and 20000.SELECT * FROM Employee WHERE Salary BETWEEN 5000 AND 20000;
15) Display the list of employee excluding job title as ‘salesman’.SELECT * FROM Employee WHERE Job != 'Salesman';
16) Display all those employees whose job title is either ‘manager’ or ‘analyst’(write by using OR & IN operator).SELECT * FROM Employee WHERE Job = 'Manager' OR Job = 'Analyst'; SELECT * FROM Employee WHERE Job IN ('Manager', 'Analyst');
17) Display the employee name & department number of all employees in dept 10,20,30& 40. **SELECT Ename, Deptno FROM Employee WHERE Deptno IN (10, 20, 30, 40);**
18) Display the employee number, name, job & commission of all employees who do not get any commission. **SELECT Empno, Ename, Job, Commission FROM Employee WHERE Commission IS NULL;**
19) Display the name & salary of all employees whose salary not in the range of 5000 & 10000. SELECT Ename, Salary FROM Employee WHERE Salary NOT BETWEEN 5000 AND 10000;
20) Find all names & joined date of employees whose names starts with ‘A’. SELECT Ename, Joined_date FROM Employee WHERE Ename LIKE 'A%';
21) Find all names of employees having ‘i’ as a second letter in their names. SELECT Ename FROM Employee WHERE Ename LIKE '_i%';
22) Find employee number, name of employees whose commission is not null.SELECT Empno, Ename FROM Employee WHERE Commission IS NOT NULL;
23) Display all employee information in the descending order of employee number.SELECT * FROM Employee ORDER BY Empno DESC;
24) Display the minimum, maximum, sum & average salary of each job type.SELECT Job, MIN(Salary), MAX(Salary), SUM(Salary), AVG(Salary) FROM Employee GROUP BY Job;

25) Write a query to display the number of employee with the same department.SELECT Deptno, COUNT(*) AS Num_Employees FROM Employee GROUP BY Deptno;
26) Select employee number, ename according to the annual salary in ascending order. SELECT Empno, Ename FROM Employee ORDER BY Salary;
27) Find the department number, maximum salary where the maximum salary is greater than 5000.SELECT Deptno, MAX(Salary) AS Max_Salary FROM Employee GROUP BY Deptno HAVING MAX(Salary) > 5000;
28) Find all distinct column values from employee & department table. SELECT DISTINCT * FROM Employee; SELECT DISTINCT * FROM Department;
29) Find all column values with duplicate from employee & department table.SELECT emp, COUNT(*) FROM person name, street, city GROUP BY person name, street, city HAVING COUNT(*) > 1;
30) Find all column values which are common in both employee & department table.SELECT e.Deptno FROM Employee e JOIN Department d ON e.Deptno = d.Deptno;
31) Find all distinct column values present in employee but not in department table.SELECT e.Deptno FROM Employee e LEFT JOIN Department d ON e.Deptno = d.Deptno WHERE d.Deptno IS NULL;
32) Display the number of employees in the department 30 who can earn a commission SELECT COUNT(*) FROM Employee WHERE Deptno = 30 AND Commission IS NOT NULL;

Create following tables with primary key and foreign key and solve below queries
person (driver id, name, address)
car (license, model, year)
accident (report number, date, location)
owns (driver id, license)
participated (report number, license, driver id, damage amount)

```
CREATE TABLE Person (
  driver_id INT PRIMARY KEY,
  name VARCHAR(50) UNIQUE,
  address VARCHAR(100) NOT NULL
);

CREATE TABLE Car (
  license VARCHAR(20) PRIMARY KEY,
  model VARCHAR(50),
  year INT
);

CREATE TABLE Accident (
  report_number INT PRIMARY KEY,
  date DATE,
  location VARCHAR(100)
);
```

```
CREATE TABLE Owns (
   driver_id INT,
   license VARCHAR(20),
   PRIMARY KEY (driver_id, license),
   FOREIGN KEY (driver_id) REFERENCES Person(driver_id),
   FOREIGN KEY (license) REFERENCES Car(license)
);

CREATE TABLE Participated (
   report_number INT,
   license VARCHAR(20),
   driver_id INT,
   damage_amount DECIMAL(10,2),
   PRIMARY KEY (report_number, license, driver_id),
   FOREIGN KEY (report_number) REFERENCES Accident(report_number),
   FOREIGN KEY (license) REFERENCES Car(license),
   FOREIGN KEY (driver_id) REFERENCES Person(driver_id)
);
```

1. Insert 5 records in each table INSERT INTO Person (driver_id, name, address) VALUES (1, 'John Smith', '123 Main St'), (2, 'Jane Doe', '456 Elm St'), (3, 'Mike Johnson', '789 Oak St'), (4, 'Emily Brown', '101 Pine St'), (5, 'David Lee', '222 Cedar St'); INSERT INTO Car (license, model, year) VALUES ('A123', 'Toyota Camry', 2010), ('B456', 'Honda Civic', 2015), ('C789', 'Ford Mustang', 2018), ('D101', 'Chevrolet Silverado', 2020), ('E222', 'Mazda CX-5', 2017);INSERT INTO Accident (report_number, date, location) VALUES (1001, '2008-01-01', 'Los Angeles'), (1002, '2009-05-15', 'New York'), (1003, '2010-12-25', 'Chicago'), (1004, '2011-08-10', 'Miami'), (1005, '2012-02-20', 'Dallas');

   INSERT INTO Owns (driver_id, license) VALUES (1, 'A123'), (2, 'B456'), (3, 'C789'), (4, 'D101'), (5, 'E222'); INSERT INTO Participated (report_number, license, driver_id, damage_amount) VALUES (1001, 'A123', 1, 1000), (1002, 'B456', 2, 2000), (1003, 'C789', 3, 3000), (1004, 'D101', 4, 4000), (1005, 'E222', 5, 5000);
2. Add unique key for name of person. ALTER TABLE Person ADD UNIQUE (name);
3. Add not null constraints for address. ALTER TABLE Person MODIFY address NOT NULL;
4. Find the total number of people who owned cars that were involved in accidents in 2009.

   ```
   SELECT COUNT(DISTINCT p.driver_id)
   FROM Person p
   JOIN Owns o ON p.driver_id = o.driver_id
   JOIN Participated pa ON o.license = pa.license
   JOIN Accident a ON pa.report_number = a.report_number
   WHERE a.date BETWEEN '2009-01-01' AND '2009-12-31';
   ```
5. Add a new accident to the database; assume any values for required attributes. INSERT INTO Accident (report_number, date, location) VALUES (1006, '2023-11-06', 'San Francisco');
6. Delete the Mazda belonging to “John Smith” DELETE FROM Owns WHERE driver_id = (SELECT driver_id FROM Person WHERE name = 'John Smith') AND license = (SELECT license FROM Car WHERE model = 'Mazda CX-5');

Design at least 10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.

Employee(employee_name,street,city)
Works(employee_name,company_name,salary)
Company(company_name,city)
Manages(employee_name,manager_name)

```
CREATE TABLE Employee (
   employee_name VARCHAR(50) PRIMARY KEY,
   street VARCHAR(100),
   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 Company (
   company_name VARCHAR(50) PRIMARY KEY,
   city VARCHAR(50)
);

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)
);
```

1. Find the names of employees who work for First Bank Coorporation.SELECT employee_name FROM Works WHERE company_name = 'First Bank Coorporation';

2. Find the names and cities of residence of all employees who work for First Bank Coorporation

```
SELECT e.employee_name, e.city
FROM Employee e
JOIN Works w ON e.employee_name = w.employee_name
WHERE w.company_name = 'First Bank Coorporation';
```

3. Find the names, street addresses, and cities of residence of all employees who work for First Bank Coorporation and earn more than $10000. SELECT e.employee_name, e.street, e.city

FROM Employee e
JOIN Works w ON e.employee_name = w.employee_name
WHERE w.company_name = 'First Bank Coorporation' AND w.salary > 10000;
4.Find all employees in the database who lives in the same cities as tha companies for which they work. SELECT e.employee_name
FROM Employee e
JOIN Works w ON e.employee_name = w.employee_name
JOIN Company c ON w.company_name = c.company_name
WHERE e.city = c.city;
5.Find all employees in the database who lives in the same cities and on the same streets as do their manager. SELECT e.employee_name
FROM Employee e
JOIN Manages m ON e.employee_name = m.employee_name
WHERE e.street = m.street AND e.city = m.city;
6.Find all employees in the database who do not work for First Bank Coorporation SELECT e.employee_name
FROM Employee e
JOIN Works w ON e.employee_name = w.employee_name
WHERE w.company_name != 'First Bank Coorporation';
7.Find all employees in the database who earn more than each employee of Small Bank Coorporation SELECT e.employee_name
FROM Employee e
JOIN Works w ON e.employee_name = w.employee_name
WHERE w.salary > (SELECT MAX(salary) FROM Works WHERE company_name = 'Small Bank Coorporation');
8.Assume that the company is may be located in several cities. Find all companies located in every city in which Small Bank Coorporation is located. SELECT c.company_name
FROM Company c
WHERE c.city IN (SELECT city FROM Company WHERE company_name = 'Small Bank Coorporation');
9.Find all employees who earn more than the average salary of all employees of their companies. SELECT e.employee_name
FROM Employee e
JOIN Works w ON e.employee_name = w.employee_name
WHERE w.salary > (SELECT AVG(salary) FROM Works WHERE company_name = w.company_name);
10.Find the company that has the most employees. SELECT company_name, COUNT(*) AS num_employees
FROM Works
GROUP BY company_name
ORDER BY num_employees DESC
LIMIT 1;

11.Find the company that has the smallest payroll. SELECT company_name, SUM(salary) AS total_salary
FROM Works
GROUP BY company_name
ORDER BY total_salary ASC
LIMIT 1;
12.Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Coorporation. SELECT company_name
FROM Works
GROUP BY company_name
HAVING AVG(salary) > (SELECT AVG(salary) FROM Works WHERE company_name = 'First Bank Coorporation');
13.Modify the database so that “Jones” now lives in Newtown.UPDATE Employee
SET street = 'Newtown'
WHERE employee_name = 'Jones';
14.Give all employees of First Bank Coorporation” a 10% raise UPDATE Works
SET salary = salary * 1.1
WHERE company_name = 'First Bank Coorporation';
15.Delete all tuples in the “Works” relation for employees of “Small Bank Coorporation”.DELETE FROM Works
WHERE company_name = 'Small Bank Coorporation';

PL/SQL

Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. Suggested Problem statement: Consider Tables: 1. Borrower(Roll_no, Name, DateofIssue, NameofBook, Status) 2. Fine(Roll_no,Date,Amt)
Accept Roll_no and NameofBook from user.
· Check the number of days (from date of issue).
· If days are between 15 to 30 then fine amount will be Rs 5per day.
· If no. of days>30, per day fine will be Rs 50 per day and for days less than 30, Rs. 5 per day.
· After submitting the book, status will change from I to R.
· If condition of fine is true, then details will be stored into fine table.
· Also handles the exception by named exception handler or user define exception handler.

Write a PL/SQL block to calculate factorial. Use Exception Handling for negative number

```
SET SERVEROUT ON
SET VERIFY OFF
/*
CREATE TABLE borrower(roll_no NUMBER , name VARCHAR2(25), dateofissue
DATE,name_of_book VARCHAR2(25), status VARCHAR2(20));
CREATE TABLE fine(roll_no NUMBER,date_of_return DATE,amt NUMBER);
INSERT INTO borrower VALUES(45,'ASHUTOSH',TO_DATE('01-08-2022','DD-MM-
```

```
YYYY'),'HARRY POTTER','PENDING');
INSERT INTO borrower VALUES(46,'ARYAN',TO_DATE('15-08-2022','DD-MM-
YYYY'),'DARK MATTER','PENDING');
INSERT INTO borrower VALUES(47,'ROHAN',TO_DATE('24-08-2022','DD-MM-
YYYY'),'SILENT HILL','PENDING');
INSERT INTO borrower VALUES(48,'SANKET',TO_DATE('26-08-2022','DD-MM-
YYYY'),'GOD OF WAR','PENDING');
INSERT INTO borrower VALUES(49,'SARTHAK',TO_DATE('09-09-2022','DD-MM-
YYYY'),'SPIDER-MAN','PENDING');
*/
DECLARE
i_roll_no NUMBER;
name_of_book VARCHAR2(25);
no_of_days NUMBER;
return_date DATE := TO_DATE(SYSDATE,'DD-MM-YYYY');
temp NUMBER;
doi DATE;
fine NUMBER;
BEGIN
i_roll_no := &i_roll_no;
name_of_book := '&nameofbook';
--dbms_output.put_line(return_date);
SELECT to_date(borrower.dateofissue,'DD-MM-YYYY') INTO doi FROM
borrower WHERE borrower.roll_no = i_roll_no AND borrower.name_of_book =
name_of_book;
no_of_days := return_date-doi;
dbms_output.put_line(no_of_days);
IF (no_of_days >15 AND no_of_days <=30) THEN
fine := 5*no_of_days;
ELSIF (no_of_days>30 ) THEN
temp := no_of_days-30;
fine := 150 + temp*50;
END IF;
dbms_output.put_line(fine);
INSERT INTO fine VALUES(i_roll_no,return_date,fine);
UPDATE borrower SET status = 'RETURNED' WHERE borrower.roll_no =
i_roll_no;
END;
/
/*
---------------------------------
```

Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area.

Code:

```
SQL> create table circlearea(radius int,area int);

Table created.

SQL> select * from circlearea;

no rows selected

SQL> edit a1.sql

declare
r1 circlearea.radius%type;
cir_area int;
pi int;
begin
r1:=5;
pi:=3.14;
for r1 in 5..9
loop
cir_area:=pi*(r1*r1);
insert into circlearea values(r1,cir_area);
end loop;
end;
/
SQL> @a1.sql

PL/SQL procedure successfully completed.

SQL> select * from circlearea;

    RADIUS       AREA
---------- ----------
         5         75
         6        108
         7        147
         8        192
         9        243
```

Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created

table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped. Frame the separate problem statement for writing PL/SQL block to implement all types

Note: You have to execute any two type of cursor which is specified in your problem statement or a merging of two tables using parameterized cursor.

Create the database and use it

```
CREATE DATABASE assi7;
USE assi7;
-- Create the old_roll and new_roll tables
CREATE TABLE old_roll (roll INT, name VARCHAR(10));
CREATE TABLE new_roll (roll INT, name VARCHAR(10));
-- Insert data into the old_roll table
INSERT INTO old_roll VALUES (4, 'D');
INSERT INTO old_roll VALUES (3, 'BCD');
INSERT INTO old_roll VALUES (1, 'BC');
INSERT INTO old_roll VALUES (5, 'BCH');
-- Insert data into the new_roll table with new names
INSERT INTO new_roll VALUES (2, 'Yash');
INSERT INTO new_roll VALUES (5, 'Varad'); -- Duplicate, will be skipped
INSERT INTO new_roll VALUES (1, 'Gauri'); -- Duplicate, will be skipped
INSERT INTO new_roll VALUES (6, 'Shravani');
INSERT INTO new_roll VALUES (7, 'Vishwajeet');
-- Display the initial data
SELECT * FROM old_roll;
SELECT * FROM new_roll;
-- Define the procedure to merge data
DELIMITER $$
CREATE PROCEDURE roll_list()
BEGIN
 DECLARE oldrollnumber INT;
 DECLARE oldname VARCHAR(10);
 DECLARE newrollnumber INT;
 DECLARE newname VARCHAR(10);
 DECLARE done INT DEFAULT FALSE;

 -- Declare cursors for old_roll and new_roll
 DECLARE c1 CURSOR FOR SELECT roll, name FROM old_roll;
 DECLARE c2 CURSOR FOR SELECT roll, name FROM new_roll;

 -- Continue handler for cursor not found
 DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
Open the new_roll cursor
 OPEN c2;
```

```
loop1: LOOP
 -- Fetch new roll data
 FETCH c2 INTO newrollnumber, newname;
 IF done THEN
 LEAVE loop1;
 END IF;
 SET done = FALSE; -- Reset done for the old_roll cursor

 -- Open the old_roll cursor
 OPEN c1;

 loop2: LOOP
 -- Fetch old roll data
 FETCH c1 INTO oldrollnumber, oldname;
 IF done THEN
 LEAVE loop2;
 END IF;
 -- Check for duplicates
 IF oldrollnumber = newrollnumber THEN
 LEAVE loop2; -- Skip if duplicate found
 END IF;
 END LOOP;

 -- Insert if no duplicates were found
 IF done THEN
 INSERT INTO old_roll (roll, name) VALUES (newrollnumber, newname);
 END IF;
 CLOSE c1; -- Close old_roll cursor after processing
 END LOOP;
 CLOSE c2; -- Close new_roll cursor after processing
END $$
DELIMITER ;
-- Call the procedure to merge data
CALL roll_list();
-- Display the merged old_roll table
SELECT * FROM old_roll;
```

Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is · <=1500 and marks>=990 then student will be placed in Distinction category · if marks scored are between 989 and 900 category is First Class, · if marks 899 and 825 category is Higher Second Class. Write a PL/SQL block for using procedure created with above requirement. 1. Stud_Marks(Roll, Name, Total_marks) 2. Result(Roll, Name, Class)

Frame the separate problem statement for writing PL/SQL Stored Procedure and function, inline with above statement. The problem statement should clearly state the requirements

```
CREATE DATABASE stud;
USE stud;
CREATE TABLE stud_marks (
 roll_no INTEGER PRIMARY KEY,
 name VARCHAR(20),
 total_marks INTEGER
);
CREATE TABLE result (
 roll_no INTEGER,
 class VARCHAR(20),
 CONSTRAINT xyz FOREIGN KEY (roll_no) REFERENCES stud_marks(roll_no)
);
INSERT INTO stud_marks (roll_no, name, total_marks) VALUES
(1, 'Mitali', 920),
(2, 'Divya', 1150),
(3, 'Rushali', 950),
(4, 'Poojs', 840),
(5, 'Rutuja', 1000),
(6, 'Rani', 860);
DELIMITER //
CREATE PROCEDURE proc_Grade(IN roll INTEGER)
BEGIN
 DECLARE m INTEGER;
 DECLARE c VARCHAR(20);
 SELECT total_marks INTO m FROM stud_marks WHERE roll_no = roll;
 IF m >= 990 AND m <= 1500 THEN
 SET c = 'Distinction';
 ELSEIF m >= 900 AND m <= 989 THEN
 SET c = 'First Class';
 ELSEIF m >= 825 AND m <= 899 THEN
 SET c = 'Higher Second Class';
 END IF;
 INSERT INTO result (roll_no, class) VALUES (roll, c);
END //
DELIMITER ;
CALL proc_Grade(1);
CALL proc_Grade(2);
CALL proc_Grade(3);
CALL proc_Grade(4);
```

```
CALL proc_Grade(5);
CALL proc_Grade(6);
SELECT * FROM result;
DELIMITER //
CREATE FUNCTION disp_grade2(roll_no INTEGER) RETURNS VARCHAR(20)
BEGIN
 DECLARE m INTEGER;
 DECLARE c VARCHAR(20);
 SELECT total_marks INTO m FROM stud_marks WHERE roll_no = roll_no;
 IF m >= 990 AND m <= 1500 THEN
 SET c = 'Distinction';
 ELSEIF m >= 900 AND m <= 989 THEN
 SET c = 'First Class';
 ELSEIF m >= 825 AND m <= 899 THEN
 SET c = 'Higher Second Class';
 END IF;
 RETURN c;
END //
DELIMITER ;
SELECT disp_grade2(1);
SELECT disp_grade2(2);
SELECT disp_grade2(3);
SELECT disp_grade2(4);
SELECT disp_grade2(5);
SELECT disp_grade2(6);
```

Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers). Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_ Audit table. Frame the problem statement for writing Database Triggers of all types, in-line with above statement. The problem statement should clearly state the requirements.

Step 1: Create the Library Table

```
CREATE TABLE Library (
 id INT PRIMARY KEY,
 name VARCHAR(50),
 author VARCHAR(50),
 year_published INT
);
-- Step 2: Create the Library_Audit Table
CREATE TABLE Library_Audit (
 audit_id INT AUTO_INCREMENT PRIMARY KEY,
 id INT,
 name VARCHAR(50),
```

```
 author VARCHAR(50),
 year_published INT,
 action VARCHAR(10),
 change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Step 3: Create Row-Level Triggers
-- Trigger to log old values before an update
DELIMITER $$
CREATE TRIGGER before_update_library
BEFORE UPDATE ON Library
FOR EACH ROW
BEGIN
 INSERT INTO Library_Audit (id, name, author, year_published, action)
 VALUES (OLD.id, OLD.name, OLD.author, OLD.year_published, 'UPDATE');
END$$
DELIMITER ;
-- Trigger to log old values before a delete
DELIMITER $$
CREATE TRIGGER before_delete_library
BEFORE DELETE ON Library
FOR EACH ROW
BEGIN
 INSERT INTO Library_Audit (id, name, author, year_published, action)
 VALUES (OLD.id, OLD.name, OLD.author, OLD.year_published, 'DELETE');
End
/
DELIMITER ;
-- Step 4: Create Statement-Level Triggers (Optional)
-- (Typically not necessary for auditing)
-- Trigger to log after an update (optional)
DELIMITER $$
CREATE TRIGGER after_update_library
AFTER UPDATE ON Library
BEGIN
 -- This could log additional information if desired
END$$
DELIMITER ;
-- Trigger to log after a delete (optional)
DELIMITER $$
CREATE TRIGGER after_delete_library
AFTER DELETE ON Library
BEGIN
 -- This could log additional information if desired
```

```
END$$
DELIMITER ;
-- Step 5: Test the Triggers
-- Insert example records into Library
INSERT INTO Library VALUES (1, 'Book A', 'Author A', 2020);
INSERT INTO Library VALUES (2, 'Book B', 'Author B', 2021);
-- Update a record
UPDATE Library SET name = 'Updated Book A' WHERE id = 1;
-- Delete a record
DELETE FROM Library WHERE id = 2;
-- Step 6: Check the Audit Log
SELECT * FROM Library_Audit;
-- Display current state of the Library table
SELECT * FROM Library;
```

MONGODB

Create database Employee.
2.Create collection emp1 using createCollection method.
3.Insert 4 to 5 documents in emp1 collection.(eno,ename,address,sal)
4.display all documents.
5.Insert document by creating own object id.
6.Display all employess having salary greater than 5000
7.Display all employess having salary less than 15000
8.Display all employess having salary greater than 10000 and less than 20000.
9.Update sal of all employee with 10%
10.Delete employee having salary less than 5000.
11.Rename collection emp1 with employee1.
12.Display employee whose name start with n.
13.sort name in ascending order.
14.Create a new database with name Employee1.
15.Drop employee1 database.
16.Create collection emp1 using insert method.
17.Drop collection emp1.

```
// 1. Create a database named "Employee"
use Employee

// 2. Create a collection named "emp1"
db.createCollection("emp1")

// 3. Insert 4-5 documents into the "emp1" collection
db.emp1.insertMany([
    { eno: 101, ename: "Alice", address: "New York", sal: 12000 },
    { eno: 102, ename: "Bob", address: "London", sal: 8000 },
    { eno: 103, "ename": "Charlie", "address": "Paris", "sal": 15000 },
    { eno: 104, "ename": "David", "address": "Berlin", "sal": 4000 },
```

```
    { eno: 105, "ename": "Eve", "address": "Tokyo", "sal": 20000 }
])

// 4. Display all documents
db.emp1.find()

// 5. Insert a document with a custom ObjectId
db.emp1.insertOne({ _id: ObjectId("64b9d7678273a24434567890"), eno: 106, ename: "Frank",
address: "Sydney", sal: 18000 })

// 6. Display employees with salary greater than 5000
db.emp1.find({ sal: { $gt: 5000 } })

// 7. Display employees with salary less than 15000
db.emp1.find({ sal: { $lt: 15000 } })

// 8. Display employees with salary between 10000 and 20000
db.emp1.find({ sal: { $gt: 10000, $lt: 20000 } })

// 9. Update salary of all employees by 10%
db.emp1.updateMany({}, { $inc: { sal: 1000 } })

// 10. Delete employees with salary less than 5000
db.emp1.deleteMany({ sal: { $lt: 5000 } })

// 11. Rename the collection
db.emp1.renameCollection("employee1")

// 12. Display employees whose name starts with 'n'
db.employee1.find({ ename: /^n/i })

// 13. Sort employees by name in ascending order
db.employee1.find().sort({ ename: 1 })

// 14. Create a new database named "Employee1"
use Employee1

// 15. Drop the "employee1" database
db.dropDatabase()

// 16. Create a collection named "emp1" using the insert method
db.emp1.insertOne({ eno: 101, ename: "Alice", address: "New York", sal: 12000 })

// 17. Drop the "emp1" collection
db.emp1.drop()
```

Create a hypothetical sports club database that contains a users collection that tracks the user's join dates, sport preferences, and stores these data in documents and perform the following operation on it using aggregation.

1. Create index on sport preferences.

```
use sports_club

db.createCollection("users")

db.users.insertMany([
  { name: "Alice", joinDate: ISODate("2023-01-15"), preferences: ["tennis", "swimming"] },
  { name: "Bob", joinDate: ISODate("2023-04-02"), preferences: ["football", "basketball"] },
  { name: "Charlie", joinDate: ISODate("2023-07-10"), preferences: ["tennis", "swimming"] },
  { name: "David", joinDate: ISODate("2023-10-25"), preferences: ["basketball", "volleyball"] },
  { name: "Eve", joinDate: ISODate("2023-02-20"), preferences: ["swimming", "yoga"] }
])
```

2. Create index on join dates and preferences.

```
// Index on sport preferences
db.users.createIndex({ preferences: 1 })

// Index on joinDate and preferences
db.users.createIndex({ joinDate: 1, preferences: 1 })
```

3. Returns user names in upper case and in alphabetical order.

```
db.users.aggregate([
  { $project: { name: { $toUpper: "$name" } } },
  { $sort: { name: 1 } }
])
```

4. Returns user names sorted by the month they joined.

```
db.users.aggregate([
  { $project: { name: 1, monthJoined: { $month: "$joinDate" } } },
  { $sort: { monthJoined: 1, name: 1 } }
])
```

5. Show how many people joined each month of the year.

```
db.users.aggregate([
  { $group: { _id: { $month: "$joinDate" }, count: { $sum: 1 } } },
  { $sort: { _id: 1 } }
])
```

Create database employee and create collection computer

Define a map function to emit the "total" key and the "Salary" value for each document.

Define a reduce function to sum the salaries associated with the "total" key.Execute the map-reduce operation on the "Computer" collection, specifying an output collection where the results will be stored.

```
use employee
db.createCollection("computer")
db.computer.insertMany([
   { _id: 1, name: "John Doe", salary: 5000 },
   { _id: 2, name: "Jane Smith", salary: 6000 },
   { _id: 3, name: "Mike Johnson", salary: 4000 }
])
var map = function() {
   emit("total", this.salary);
};

var reduce = function(key, values) {
   var total = 0;
   for (var i = 0; i < values.length; i++) {
      total += values[i];
   }
   return total;
};
db.computer.mapReduce(
   map,
   reduce,
   { out: "total_salary" }
)
```