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)

1) **Create view with the employee_name, company_name by using above tables.**

```
CREATE VIEW EmployeeCompanyView AS
SELECT e.employee_name, w.company_name
FROM Employee e
JOIN Works w ON e.employee_name = w.employee_name;
```

2) **Create index for employee & participated table.**

```
-- Index on Employee table
CREATE INDEX idx_employee_name ON Employee (employee_name);

-- Index on Participated table
CREATE INDEX idx_participated_driver_id ON Participated (driver_id);
```

3) **Create sequence for person & insert 4 records using sequence.**

```
-- Sequence for Person table
CREATE SEQUENCE person_seq START WITH 1 INCREMENT BY 1;

-- Insert records into Person table
INSERT INTO Person (driver_id, name, address) VALUES (person_seq.NEXTVAL,
'Alice', '123 Elm St');
INSERT INTO Person (driver_id, name, address) VALUES (person_seq.NEXTVAL,
'Bob', '456 Oak St');
INSERT INTO Person (driver_id, name, address) VALUES (person_seq.NEXTVAL,
'Charlie', '789 Maple Ave');
```

```
INSERT INTO Person (driver_id, name, address) VALUES (person_seq.NEXTVAL, 'Diana', '101 Pine Rd');
```

4) **Create the synonym for table participated & company. Display the record using this table. Update the record using the synonym tables.**

```
-- Synonym for Participated table
CREATE SYNONYM Syn_Participated FOR Participated;
-- Synonym for Company table
CREATE SYNONYM Syn_Company FOR Company;

-- Display records using the synonyms
SELECT * FROM Syn_Participated;
SELECT * FROM Syn_Company;

-- Example update using synonyms
UPDATE Syn_Participated
SET damage_amount = 500.00
WHERE driver_id = 1 AND report_no = 101;

UPDATE Syn_Company
SET city = 'New City'
WHERE company_name = 'TechCorp';
```

5) **Drop index from employee table.**

```
DROP INDEX idx_employee_name;
```

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

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

1) **Create view with the Customer name, customer street by using above tables.**

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

-- Table: Works
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)
);
-- Table: Company
CREATE TABLE Company (
    company_name VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50)
);
-- Table: Branch
CREATE TABLE Branch (
    branch_name VARCHAR(50) PRIMARY KEY,
    branch_city VARCHAR(50),
    assets DECIMAL(15, 2)
);

-- Table: Customer
CREATE TABLE Customer (
    customer_name VARCHAR(50) PRIMARY KEY,
    customer_street VARCHAR(100),
    customer_city VARCHAR(50)
);
-- Table: Loan
CREATE TABLE Loan (
    loan_number INT PRIMARY KEY,
    branch_name VARCHAR(50),
    amount DECIMAL(10, 2),
    FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);
-- Table: Borrower
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)
);

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

-- Table: Depositor
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)
);
CREATE VIEW CustomerInfoView AS
SELECT customer_name, customer_street
FROM Customer;
```

2) Create index for loan number.

```
CREATE INDEX idx_loan_number ON Loan (loan_number);
```

3) Create sequence for account & insert 4 records using sequence.

```
-- Sequence for Account table
CREATE SEQUENCE account_seq START WITH 1 INCREMENT BY 1;
-- Insert records into Account table
INSERT INTO Account (account_number, branch_name, balance) VALUES
(account_seq.NEXTVAL, 'MainBranch', 1000.00);
INSERT INTO Account (account_number, branch_name, balance) VALUES
(account_seq.NEXTVAL, 'DowntownBranch', 1500.00);
INSERT INTO Account (account_number, branch_name, balance) VALUES
(account_seq.NEXTVAL, 'UptownBranch', 2000.00);
INSERT INTO Account (account_number, branch_name, balance) VALUES
(account_seq.NEXTVAL, 'SuburbanBranch', 2500.00);
```

4) **Create the synonym for table depositor & borrower. Display the record using this table. Update the record using the synonym tables.**

```
- Synonym for Depositor table
CREATE SYNONYM Syn_Depositor FOR Depositor;

-- Synonym for Borrower table
CREATE SYNONYM Syn_Borrower FOR Borrower;
```

```
-- Display records using the synonyms
SELECT * FROM Syn_Depositor;
SELECT * FROM Syn_Borrower;
-- Example update using synonyms
UPDATE Syn_Depositor
SET account_number = 102
WHERE customer_name = 'Alice';

UPDATE Syn_Borrower
SET loan_number = 203
WHERE customer_name = 'Bob';
```

5) **Drop synonym for depositor table.**

```
DROP SYNONYM Syn_Depositor;
```

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(50),
   location VARCHAR(50)
);
```

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');
INSERT INTO Department (deptno, dname, location) VALUES (20, 'Research', 'Pune');
INSERT INTO Department (deptno, dname, location) VALUES (30, 'Sales', 'Nashik');
INSERT INTO Department (deptno, dname, location) VALUES (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(50),
    job VARCHAR(50),
    mgr INT,
    joined_date DATE,
    salary DECIMAL(10, 2),
    commission DECIMAL(10, 2),
    deptno INT,
    address VARCHAR(50),
    FOREIGN KEY (deptno) REFERENCES Department(deptno)
);
```

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	**Avinash pawar**	**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	**Pushkar deshpande**	**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 VALUES (1001, 'Nilesh Joshi', 'Clerk', 1005, '1995-12-17',
2800, 600, 20, 'Nashik');
INSERT INTO Employee VALUES (1002, 'Avinash Pawar', 'Salesman', 1003, '1996-02-
20', 5000, 1200, 30, 'Nagpur');
INSERT INTO Employee VALUES (1003, 'Amit Kumar', 'Manager', 1004, '1986-04-02',
2000, NULL, 30, 'Pune');
INSERT INTO Employee VALUES (1004, 'Nitin Kulkarni', 'President', NULL, '1986-
04-19', 50000, NULL, 10, 'Mumbai');
INSERT INTO Employee VALUES (1005, 'Niraj Sharma', 'Analyst', 1003, '1998-12-03',
12000, NULL, 20, 'Satara');
INSERT INTO Employee VALUES (1006, 'Pushkar Deshpande', 'Salesman', 1003,
'1996-09-01', 6500, 1500, 30, 'Pune');
INSERT INTO Employee VALUES (1007, 'Sumit Patil', 'Manager', 1004, '1991-05-01',
25000, NULL, 20, 'Mumbai');
```

INSERT INTO Employee VALUES (1008, 'Ravi Sawant', 'Analyst', 1007, '1995-11-17', 10000, NULL, NULL, 'Amaravati');

6) **Write a query to display employee information. Write a name of column explicitly.**
   SELECT empno AS "Employee Number", ename AS "Employee Name", job AS "Job Title", salary AS "Salary", commission AS "Commission", deptno AS "Department Number" FROM Employee;
7) **Add column phone number in the employee table and add not null constraints.**
   ALTER TABLE Employee ADD phone_number VARCHAR(15) 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 AS SELECT * FROM Department WHERE 1=0;
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) AS min_salary, MAX(salary) AS max_salary, SUM(salary) AS total_salary, AVG(salary) AS 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 employee_count FROM Employee GROUP BY deptno;

26) **Select employee number, ename according to the annual salary in ascending order.**
SELECT empno, ename, (salary * 12) AS annual_salary FROM Employee ORDER BY annual_salary ASC;

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 deptno, dname, location FROM Department;
SELECT DISTINCT empno, ename, job, deptno FROM Employee;

29) **Find all column values with duplicate from employee & department table.**
SELECT deptno, COUNT() FROM Department GROUP BY deptno HAVING COUNT() > 1;
SELECT empno, COUNT() FROM Employee GROUP BY empno HAVING COUNT() > 1;

30) **Find all column values which are common in both employee & department table.**
SELECT Employee.deptno FROM Employee JOIN Department ON Employee.deptno = Department.deptno;

31) **Find all distinct column values present in employee but not in department table.**
SELECT DISTINCT deptno FROM Employee WHERE deptno NOT IN (SELECT deptno FROM Department);

32) **Display the number of employees in the department 30 who can earn a commission**
SELECT COUNT(*) AS commission_earners 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)

1. **Insert 5 records in each table**

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

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

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

-- Table: Owns
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)
);

-- Table: Participated
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)
);
-- Insert records into Person table
INSERT INTO Person (driver_id, name, address) VALUES (1, 'John Smith', '123 Main St');
INSERT INTO Person (driver_id, name, address) VALUES (2, 'Alice Brown', '456 Oak Ave');
INSERT INTO Person (driver_id, name, address) VALUES (3, 'Michael Lee', '789 Pine Dr');
INSERT INTO Person (driver_id, name, address) VALUES (4, 'Emily White', '101 Maple St');
INSERT INTO Person (driver_id, name, address) VALUES (5, 'David Green', '202 Birch Rd');

-- Insert records into Car table
INSERT INTO Car (license, model, year) VALUES ('ABC123', 'Toyota', 2008);
INSERT INTO Car (license, model, year) VALUES ('DEF456', 'Honda', 2009);
INSERT INTO Car (license, model, year) VALUES ('GHI789', 'Mazda', 2010);
INSERT INTO Car (license, model, year) VALUES ('JKL012', 'Ford', 2011);
INSERT INTO Car (license, model, year) VALUES ('MNO345', 'BMW', 2009);

-- Insert records into Accident table
INSERT INTO Accident (report_number, date, location) VALUES (101, '2009-06-15', 'Downtown');
INSERT INTO Accident (report_number, date, location) VALUES (102, '2009-08-20', 'Uptown');
INSERT INTO Accident (report_number, date, location) VALUES (103, '2010-01-05', 'Suburb');
INSERT INTO Accident (report_number, date, location) VALUES (104, '2011-03-22', 'City Center');
INSERT INTO Accident (report_number, date, location) VALUES (105, '2009-11-15', 'Industrial Area');

-- Insert records into Owns table
INSERT INTO Owns (driver_id, license) VALUES (1, 'ABC123');
INSERT INTO Owns (driver_id, license) VALUES (2, 'DEF456');
INSERT INTO Owns (driver_id, license) VALUES (3, 'GHI789');
INSERT INTO Owns (driver_id, license) VALUES (4, 'JKL012');
INSERT INTO Owns (driver_id, license) VALUES (5, 'MNO345');

-- Insert records into Participated table
INSERT INTO Participated (report_number, license, driver_id, damage_amount) VALUES (101, 'ABC123', 1, 1500.00);
```

INSERT INTO Participated (report_number, license, driver_id, damage_amount) VALUES (102, 'DEF456', 2, 2000.00);
INSERT INTO Participated (report_number, license, driver_id, damage_amount) VALUES (103, 'GHI789', 3, 1200.00);
INSERT INTO Participated (report_number, license, driver_id, damage_amount) VALUES (104, 'JKL012', 4, 2500.00);
INSERT INTO Participated (report_number, license, driver_id, damage_amount) VALUES (105, 'MNO345', 5, 1800.00);

2. **Add unique key for name of person.**
   Add Unique Key for name in Person Table (already added during creation).
3. **Add not null constraints for address.**
   Add NOT NULL Constraint for address (already added during creation).
4. **Find the total number of people who owned cars that were involved in accidents in 2009.**
   SELECT COUNT(DISTINCT p.driver_id) AS total_people
   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 YEAR(a.date) = 2009;
5. **Add a new accident to the database; assume any values for required attributes.**
   INSERT INTO Accident (report_number, date, location) VALUES (106, '2023-09-14', 'Green Valley');
6. **Delete the Mazda belonging to "John Smith"**

```
DELETE FROM Car
WHERE license = (SELECT o.license
              FROM Owns o
              JOIN Person p ON o.driver_id = p.driver_id
              WHERE p.name = 'John Smith' AND
                  (SELECT model FROM Car WHERE Car.license =
o.license) = 'Mazda');
```

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)

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

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 Corporation';
   ```

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 Corporation' 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 DISTINCT 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
JOIN Employee Mgr ON M.manager_name = Mgr.employee_name
WHERE E.city = Mgr.city AND E.street = Mgr.street;
```

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 Corporation';
```

7.Find all employees in the database who earn more than each employee of Small Bank Coorporation

```
SELECT E.employee_name
FROM Works E
WHERE E.salary > ALL (
   SELECT salary
   FROM Works
   WHERE company_name = 'Small Bank Corporation'
);
```

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 DISTINCT C.company_name
```

```
FROM Company C
WHERE NOT EXISTS (
   SELECT city
   FROM Company
   WHERE company_name = 'Small Bank Corporation' AND city NOT IN (
      SELECT city
      FROM Company
      WHERE company_name = C.company_name
   )
);
```

9.Find all employees who earn more than the average salary of all employees of their companies.

```
SELECT W.employee_name
FROM Works W
WHERE W.salary > (
   SELECT AVG(salary)
   FROM Works
   WHERE company_name = W.company_name
);
```

10.Find the company that has the most employees.

```
SELECT W.company_name
FROM Works W
GROUP BY W.company_name
ORDER BY COUNT(W.employee_name) DESC
LIMIT 1;
```

11.Find the company that has the smallest payroll.

```
SELECT W.company_name
FROM Works W
GROUP BY W.company_name
ORDER BY SUM(W.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 Corporation'
);
```

13.Modify the database so that "Jones" now lives in Newtown.

```
UPDATE Employee
```

```
SET city = '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 Corporation';
```

15.Delete all tuples in the “Works” relation for employees of “Small Bank Coorporation”.

```
DELETE FROM Works
WHERE company_name = 'Small Bank Corporation';
```

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

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.

```
mysql> show databases;
mysql> use renuka1
 mysql> create table old_roll(roll int primary key,name varchar(20),address varchar(20));
mysql> insert into old_roll values(1,"renuka","Dhoki"),(2,"ishwari","padal
i"),(3,"swarewupa","pune"),(4,"tejaswi","nashik")
mysql> select * from old_roll;
mysql> create table new_roll(roll int primary
key,name varchar(20),address varchar(20));
mysql> insert into new_roll
values(5,"renu","Dhoki"),(6,"ishu","padali"),(7
,"swaru","pune"),(8,"aditya","hadpsar");
mysql> select * from new_roll;
mysql> DELIMITER //
mysql> CREATE PROCEDURE newcur2()
```

```
-> BEGIN
-> DECLARE r_roll INT;
-> DECLARE r_name VARCHAR(20);
-> DECLARE r_address VARCHAR(20);
-> DECLARE exit_loop BOOLEAN DEFAULT FALSE;
-> DECLARE c1 CURSOR FOR SELECT roll, name, address FROM new_roll;
-> DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;
-> OPEN c1;
-> emp_loop: LOOP
-> FETCH c1 INTO r_roll, r_name, r_address;
-> IF exit_loop THEN
-> LEAVE emp_loop;
-> END IF;
-> IF NOT EXISTS (SELECT 1 FROM old_roll WHERE roll = r_roll) THEN
-> INSERT INTO old_roll (roll, name, address)
-> VALUES (r_roll, r_name, r_address);
-> END IF;
-> END LOOP;
-> CLOSE c1;
-> END;
-> //
mysql> DELIMITER ;
mysql> CALL newcur2();
mysql> 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 pk;
show databases;
use pk;
Create table result(Rooll_no int primary key,name varchar(20),class varchar(30));
Create procedure setclass2
(IN Rooll_no INT,IN name VARCHAR(30),IN total_marks INT)
BEGIN
-> IF total_marks <= 1500 AND total_marks >= 990 THEN
-> INSERT INTO result(Rooll_no,name,class)
-> VALUES(Rooll_no,name,'Distinction');
-> ELSEIF total_marks < 990 AND total_marks >=900 THEN
-> INSERT INTO result(Rooll_no,name,class)
-> VALUES(Rooll_no,name,'First class');
-> ELSEIF total_marks < 900 AND total_marks >=825 THEN
-> INSERT INTO result(Rooll_no,name,class)
-> VALUES(Rooll_no, name,'HIGHER Second Class');
-> END IF;
-> END //
mysql> Delimiter;
mysql>CALL setclass2(1,'Swaraj',1250);
mysql>select* from result;
mysql>CALL setclass2(2,'Karan',900);
mysql>select* from result;
mysql>CALL setclass2(3,'Swaranjali',870);
mysql>select* from result;
```

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.

```
mysql> Create database p1;
mysql> show databases;
mysql> use p1;
Database changed
mysql> create table Library(bno int(5),bname varchar(40),author varchar(20),allowed_days
int(5));
mysql> create table Library_audit(bno int(5),old_all_days int(5),new_all_days
int(5));
mysql> insert into Library values(1,'Python Programming','Sheetal',15);
mysql> insert into Library values(2,'Java Programming','Tanuja',10);
mysql> insert into Library values(2,'DBMS','Preeti',12);
mysql> insert into Library values(4,'Computer Networking','Vishal',9);
mysql> update Library set bno=3 where author='Preeti';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> insert into Library values(5,'Theory of
Comutation','Anuja',16);
mysql> select * from Library;
mysql> delimiter //
mysql> create trigger tr1
-> before update on Library
-> for each row
-> begin
-> insert into Library_audit values(new.bno,old.allowed_days,new.allowed_days);
-> end //
Query OK, 0 rows affected (0.07 sec)
mysql> update Library set allowed_days=21 where bno=1;
-> //
Query OK, 1 row affected (0.07 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> update Library set allowed_days=18 where bno=2;
-> //
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> update Library set allowed_days=25 where bno=3;
-> //
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> update Library set allowed_days=11 where bno=4;
-> //
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> update Library set allowed_days=28 where bno=5;
-> //
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> 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.

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.
2. Create index on join dates and preferences.
3. Returns user names in upper case and in alphabetical order.
4. Returns user names sorted by the month they joined.
5. Show how many people joined each month of the year.

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.