Problem Statement: -1

Account(Acc_no, branch_name,balance)
branch(branch_name,branch_city,assets_amt)
customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(Acc_no,loan_no,branch_name,amount)
Borrower(cust_name,loan_no)
Solve following query:

1. Create above tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.
2. Find the names of all branches in loan relation.
3. Find all loan numbers for loans made at Pimpri Branch with loan amount > 12000.
4. Find all customers who have a loan from bank. Find their names, loan_no and loan amount.
5. List all customers in alphabetical order who have loan from Akurdi branch.
6. Find all customers who have an account or loan or both at bank.
7. Find all customers who have both account and loan at bank.
8. Find average account balance at Pimpri branch.
9. Find the average account balance at each branch
10. Find the branches where average account balance > 12000.
11. Calculate total loan amount given by bank.

Solution:-

1. Create the tables with appropriate constraints:

```sql
-- Create the 'branch' table
CREATE TABLE branch (
    branch_name VARCHAR(255) PRIMARY KEY,
    branch_city VARCHAR(255) NOT NULL,
    assets_amt DECIMAL(10, 2) NOT NULL
);
-- Create the 'customer' table
CREATE TABLE customer (
    cust_name VARCHAR(255) PRIMARY KEY,
    cust_street VARCHAR(255),
    cust_city VARCHAR(255)
);
```

```
-- Create the 'account' table
CREATE TABLE account (
    Acc_no INT PRIMARY KEY,
    branch_name VARCHAR(255),
    balance DECIMAL(10, 2),
    FOREIGN KEY (branch_name) REFERENCES branch(branch_name)
);
-- Create the 'depositor' table
CREATE TABLE depositor (
    cust_name VARCHAR(255),
    acc_no INT,
    FOREIGN KEY (cust_name) REFERENCES customer(cust_name),
    FOREIGN KEY (acc_no) REFERENCES account(Acc_no)
);
-- Create the 'loan' table
CREATE TABLE loan (
    Acc_no INT,
    loan_no INT PRIMARY KEY,
    branch_name VARCHAR(255),
    amount DECIMAL(10, 2),
    FOREIGN KEY (Acc_no) REFERENCES account(Acc_no),
    FOREIGN KEY (branch_name) REFERENCES branch(branch_name)
);
-- Create the 'borrower' table
CREATE TABLE borrower (
    cust_name VARCHAR(255),
    loan_no INT,
    FOREIGN KEY (cust_name) REFERENCES customer(cust_name),
    FOREIGN KEY (loan_no) REFERENCES loan(loan_no)
);
```

```
-- Insert sample data into the 'branch' table
INSERT INTO branch (branch_name, branch_city, assets_amt) VALUES
    ('Pimpri', 'Pune', 1500000.00),
    ('Akurdi', 'Pune', 1200000.00),
    ('Chinchwad', 'Pune', 1800000.00);

-- Insert sample data into the 'customer' table with Indian names and addresses
INSERT INTO customer (cust_name, cust_street, cust_city) VALUES
    ('Rahul', '123 Gandhi Road', 'Pune'),
    ('Sunita', '456 Tagore Street', 'Pune'),
    ('Amit', '789 Nehru Avenue', 'Pune');

-- Insert sample data into the 'account' table
INSERT INTO account (Acc_no, branch_name, balance) VALUES
    (101, 'Pimpri', 5000.00),
    (102, 'Akurdi', 8000.00),
    (103, 'Chinchwad', 12000.00);

-- Insert sample data into the 'depositor' table
INSERT INTO depositor (cust_name, acc_no) VALUES
    ('Rahul', 101),
    ('Sunita', 102);

-- Insert sample data into the 'loan' table
INSERT INTO loan (Acc_no, loan_no, branch_name, amount) VALUES
    (103, 201, 'Pimpri', 15000.00),
    (102, 202, 'Akurdi', 10000.00),
    (101, 203, 'Chinchwad', 20000.00);
```

```
-- Insert sample data into the 'borrower' table
INSERT INTO borrower (cust_name, loan_no) VALUES
    ('Amit', 201),
    ('Rahul', 202);
```

Now that the tables are created and populated with sample data, you can execute the SQL queries to retrieve the desired information:

2. Find the names of all branches in the loan relation:

```sql
SELECT DISTINCT branch_name
FROM loan;
```

3. Find all loan numbers for loans made at Pimpri Branch with a loan amount > 12000:

```sql
SELECT loan_no
FROM loan
WHERE branch_name = 'Pimpri' AND amount > 12000;
```

4. Find all customers who have a loan from the bank. Find their names, loan_no, and loan amount:

```sql
SELECT c.cust_name, l.loan_no, l.amount
FROM customer c
INNER JOIN borrower b ON c.cust_name = b.cust_name
INNER JOIN loan l ON b.loan_no = l.loan_no;
```

5. List all customers in alphabetical order who have a loan from Akurdi branch:

```sql
SELECT c.cust_name
FROM customer c
INNER JOIN borrower b ON c.cust_name = b.cust_name
INNER JOIN loan l ON b.loan_no = l.loan_no
WHERE l.branch_name = 'Akurdi'
ORDER BY c.cust_name;
```

6. Find all customers who have an account or loan or both at the bank:

```sql
SELECT DISTINCT c.cust_name
FROM customer c
LEFT JOIN depositor d ON c.cust_name = d.cust_name
LEFT JOIN borrower b ON c.cust_name = b.cust_name
WHERE d.cust_name IS NOT NULL OR b.cust_name IS NOT NULL;
```

7. Find all customers who have both an account and a loan at the bank:

```sql
SELECT c.cust_name
FROM customer c
JOIN depositor d ON c.cust_name = d.cust_name
JOIN borrower b ON c.cust_name = b.cust_name;
```

8. Find the average account balance at the Pimpri branch:

```sql
SELECT AVG(balance) AS avg_balance
FROM account
WHERE branch_name = 'Pimpri';
```

9. Find the average account balance at each branch:

```sql
SELECT branch_name, AVG(balance) AS avg_balance
FROM account
GROUP BY branch_name;
```

10. Find the branches where the average account balance > 12000:

```sql
SELECT branch_name
FROM (SELECT branch_name, AVG(balance) AS avg_balance
    FROM account
    GROUP BY branch_name) AS avg_balances
WHERE avg_balance > 12000;
```

11. Calculate the total loan amount given by the bank:

```sql
SELECT SUM(amount) AS total_loan_amount
FROM loan;
```

Problem Statement:-2

1. 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 and900 category is first class, if marks 899 and 825 category is Higher Second Class
   Write a PL/SQL block for using procedure created with above requirement.

   Stud_Marks(name, total_marks)

   Result(Roll,Name, Class)

Solution:-

```
create table marks(roll_no int,name varchar(20),total_marks varchar(20));
create table result(roll_no int,name varchar(20),class varchar(20));
insert into marks values('1','Abhi','1400');
insert into marks values('2','piyush','980');
insert into marks values('3','hitesh','880');
insert into marks values('4','ashley','820');
insert into marks values('5','partik','740');
insert into marks values('6','patil','640');
mysql>
delimiter $$
mysql>
create procedure proc_result(in marks int,out class char(20))
begin
if(marks<1500&&marks>990)
then
set class='Distincton';
end if;
if(marks<989&&marks>890)
then
set class='First Class';
end if;
if(marks<889&&marks>825)
then
set class='Higher Second Class';
end if;
if(marks<824&&marks>750)
then
set class='Second Class';
end if;
if(marks<749&&marks>650)
then
```

```
set class='Passed';
end if;
if(marks<649)
then
set class='Fail';
end if;
end;
$$
mysql>
CREATE FUNCTION final_result3(R1 INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
DECLARE fmarks INT;
DECLARE grade VARCHAR(20);
DECLARE stud_name VARCHAR(20);
SELECT marks.total_marks, marks.name INTO fmarks, stud_name FROM marks WHERE
marks.roll_no = R1;
CALL proc_result(fmarks, @grade); -- Change the procedure name to proc_result
INSERT INTO result VALUES (R1, stud_name, @grade);
RETURN R1;
END;
$$
mysql>
select final_result3(2);
mysql>
select final_result3(3);
mysql>
select final_result3(4);
mysql>
select final_result3(5);
mysql>
select * from result;
```

Problem Statement:-3

1. Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_Roll_Call with the data available in the table O_Roll_Call. If the data in the first table already exists in the second table then that data should be skipped

1. First Create two table oldroll and newroll

```
create table oldroll(roll int,Name varchar(20));

create table newroll(roll int,Name varchar(20));

insert into newroll values(2,'dhanshree');

insert into newroll values(5,'asmita');

insert into oldroll values(2,'Hema');

insert into oldroll values(5,'Aditi');
insert into oldroll values(4,'srushti');
insert into oldroll values(5,'asmita');
select * from newroll;

select * from oldroll;
```

2. Creating procedure by using explicit cursor

```
delimiter $$

create procedure rolllist();
begin declare a int;
declare a1 varchar(10);
declare b int;
declare b1 varchar(10);
declare done int default false;
declare c1 cursor for select roll,name from oldroll; declare c2 cursor for select roll,name from
newroll; declare continue handler for not found set done=true; open c1;
open c2;
```

```
loop1:loop fetch c1 into a,a1;
if done then leave loop1;
end if;
loop2:loop fetch c2 into b,b1;
if done then insert into newroll values(a,a1);
leave loop2;
end if;
if a=b then leave loop2;
end if;
end loop;
end loop;
close c1;
close c2;
end $$
call rolllist() $$
select * from newroll $$
```

TRIGGER

```
create trigger deletedata2 after delete on oldroll for each row begin
insert into logtable(roll,Name,Date) values(old.roll,old.Name,curdate());
end;
 $$
```

Problem Statement: -4

Create following collections and Perform Mongodb CRUD Operations.

Teachers (Tname, dno, dname, experience, salary, date_of_joining)
Students(Sname, roll_no, class)

1. Find the information about all teachers alphabetically.
2. Find the information about all teachers of the computer department
3. Find the information about all teachers of computer,IT,and e&TC department
4. Find the information about all teachers of computer,IT,and E&TC department having salary greater than or equal to 10000/-
6. Find the student information having roll_no = 2 or Sname=xyz
7. Update the experience of teacher-praveen to 10 years, if the entry is not available in database consider the entry as new entry.
9. Update the department of all the teachers working in IT department to COMP
10. find the teacher's name and their experience from teachers' collection
11. Delete all the documents from teacher's collection having IT dept.
12. display with pretty () method, the first 3 documents in teacher's collection in ascending order.

Solution:

1) Insert Data

```
// Insert sample data into the Teachers collection
db.Teachers.insertMany([
  {
   Tname: "John",
   dno: 1,
   dname: "Computer",
   experience: 5,
   salary: 12000,
   date_of_joining: ISODate("2020-05-15")
  },
  {
   Tname: "Alice",
   dno: 2,
   dname: "IT",
   experience: 8,
   salary: 15000,
   date_of_joining: ISODate("2018-09-10")
  },
  {
   Tname: "Bob",
   dno: 3,
   dname: "E&TC",
   experience: 6,
   salary: 11000,
   date_of_joining: ISODate("2019-03-20")
```

```
  },
  {
   Tname: "Praveen",
   dno: 4,
   dname: "Computer",
   experience: 3,
   salary: 9000,
   date_of_joining: ISODate("2021-11-01")
  }
]);

// Insert sample data into the Students collection
db.Students.insertMany([
  {
   Sname: "Student1",
   roll_no: 1,
   class: "A"
  },
  {
   Sname: "Student2",
   roll_no: 2,
   class: "B"
  },
  {
   Sname: "Student3",
   roll_no: 3,
   class: "A"
  }
]);
```

1. **Find the information about all teachers alphabetically.**

→

```
db.Teachers.find().sort({ Tname: 1 })
```

2. **Find the information about all teachers of the computer department**

→

```
db.Teachers.find({ dname: "Computer" })
```

3. **Find the information about all teachers of computer,IT,and e&TC department**

→

```
db.Teachers.find({ dname: { $in: ["Computer", "IT", "E&TC"] } })
```

4. **Find the information about all teachers of computer,IT,and E&TC department having salary greater than or equal to 10000/-**

→

```
db.Teachers.find({
  dname: { $in: ["Computer", "IT", "E&TC"] },
  salary: { $gte: 10000 }
})
```

6. Find the student information having roll_no = 2 or Sname=xyz

→

```
db.Students.find({ $or: [{ roll_no: 2 }, { Sname: "xyz" }] })
```

7. Update the experience of teacher-praveen to 10 years, if the entry is not available in database consider the entry as new entry.

→

```
db.Teachers.update(
{ Tname: "Praveen" },
{ $set: { experience: 10 } },
{ upsert: true }
)
```

9. Update the department of all the teachers working in IT department to COMP

→

```
db.Teachers.updateMany(
{ dname: "IT" },
{ $set: { dname: "COMP" } }
)
```

10. find the teacher's name and their experience from teachers' collection

→

```
db.Teachers.find({}, { Tname: 1, experience: 1, _id: 0 })
```

11. Delete all the documents from teacher's collection having IT dept.

→

```
db.Teachers.deleteMany({ dname: "IT" })
```

12. display with pretty() method, the first 3 documents in teacher's collection in ascending order.

→

```
db.Teachers.find().limit(3).sort({ Tname: 1 }).pretty()
```

Problem Statement: -5

MongoDB Aggregation
You have been given a dataset containing details about different books. Each book has the following fields:

- title: The title of the book
- author: The author of the book
- genre: The genre of the book (e.g., Fiction, Non-Fiction, Mystery, Sci-Fi)
- price: The price of the book
- published_date: The date the book was published.

The data has been stored in a MongoDB collection named books.

Using the MongoDB aggregation framework, perform the following tasks:

1. Find the average price of all books.
2. Find the count of books in each genre.
3. For each genre, find the most expensive book.
4. Find the authors who have written maximum books.
5. Sort the books by published_date in descending order.
6. Sort the price in ascending order.
7. create an index on title of the book and describe the index details

Solution:-

1) First Insert Data

```
db.books.insertMany([
  {
   title: "The Great Gatsby",
   author: "F. Scott Fitzgerald",
   genre: "Fiction",
   price: 12.99,
   published_date: ISODate("1925-04-10")
  },
  {
   title: "To Kill a Mockingbird",
   author: "Harper Lee",
   genre: "Fiction",
   price: 10.99,
   published_date: ISODate("1960-07-11")
  },
  {
   title: "The Catcher in the Rye",
   author: "J.D. Salinger",
   genre: "Fiction",
   price: 9.99,
   published_date: ISODate("1951-07-16")
  },
```

```
{
 title: "The Da Vinci Code",
 author: "Dan Brown",
 genre: "Mystery",
 price: 14.99,
 published_date: ISODate("2003-03-18")
},
{
 title: "Dune",
 author: "Frank Herbert",
 genre: "Sci-Fi",
 price: 15.99,
 published_date: ISODate("1965-06-01")
},
{
 title: "Sapiens: A Brief History of Humankind",
 author: "Yuval Noah Harari",
 genre: "Non-Fiction",
 price: 16.99,
 published_date: ISODate("2011-04-11")
},
{
 title: "The Hobbit",
 author: "J.R.R. Tolkien",
 genre: "Fantasy",
 price: 11.99,
 published_date: ISODate("1937-09-21")
}])
```

1. **Find the average price of all books.**

→

```
db.books.aggregate([
  {
   $group: {
    _id: null,
    averagePrice: { $avg: "$price" }
   }
  }
])
```

2. **Find the count of books in each genre.**

→

```
db.books.aggregate([
  {
   $group: {
    _id: "$genre",
    count: { $sum: 1 }
   }
  }
])
```

3. **For each genre, find the most expensive book.**

→

```
db.books.aggregate([
  {
   $group: {
    _id: "$genre",
    maxPrice: { $max: "$price" }
   }
  }
])
```

4. **Find the authors who have written maximum books.**

→

```
 db.books.aggregate([
   {
    $group: {
      _id: "$author",
     bookCount: { $sum: 1 }
    }
   },
   {
    $sort: { bookCount: -1 }
   },
   {
    $limit: 1
   }
 ])
```

5. **Sort the books by published_date in descending order.**

→

```
db.books.aggregate([
  {
   $sort: { published_date: -1 }
  }
])
```

6. **Sort the price in ascending order.**

→

```
db.books.aggregate([
  {
   $sort: { price: 1 }
  }
])
```

7. **create an index on title of the book and describe the index details**

→

```
db.books.createIndex({ title: 1 })

db.books.getIndexes()
```

Problem Statement: -6

A retail company maintains a MongoDB collection named customer. Each document in this collection represents a purchase and contains fields such as cid (Customer ID), cname (Customer Name), amount (Amount spent on product purchase), and product_name (Product Name).

Implement a MapReduce function in MongoDB to analyze the customer collection and produce a summarized report that displays **the total amount spent by each customer on product** purchases.

Solution:

```
db.customer.insertMany([
  {
    cid: 1,
    cname: "Customer A",
    amount: 100,
    product_name: "Product X"
  },
  {
    cid: 1,
    cname: "Customer A",
    amount: 50,
    product_name: "Product Y"
  },
  {
    cid: 2,
    cname: "Customer B",
    amount: 75,
    product_name: "Product X"
  },
  {
    cid: 2,
    cname: "Customer B",
    amount: 120,
    product_name: "Product Z"
  },
  // Add more customer purchase records here
]);
```

1) Create Function

```
var mapFunction = function() {
  emit(this.cid, this.amount);
};
```

2) **Next, create a reduce function that calculates the sum of all the "amount" values for each customer (key).**

```
var reduceFunction = function(key, values) {
  return Array.sum(values);
};
```

3) **Now, you can run the MapReduce operation on the "customer" collection using the map and reduce functions you've defined. Make sure to specify an output collection where the results will be stored. In this example, the output collection is named "customer_summary."**

```
db.customer.mapReduce(
  mapFunction,
  reduceFunction,
  { out: "customer_summary" }
);
```

4) **After running the MapReduce operation, you can query the "customer_summary" collection to get the summarized report. This collection will contain documents with the customer ID (cid) as the key and the total amount spent as the value.**
   Here's a sample query to retrieve the summarized report:

```
db.customer_summary.find().forEach(function(doc) {
  print("Customer ID: " + doc._id + ", Total Amount Spent: " + doc.value);
});
```

Problem Statement: -7

Unnamed PL/SQLcode block: Use of Control structure and Exception handling is mandatory.

Suggested Problem statement:

Consider Tables:

1. Borrower (Roll_no, Name, Date_of_Issue, Name_of_Book, Status)
2. Fine (Roll_no, Date, Amt)

- Accept Roll_no and Name_of_Book 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.

Solution:

```
mysql> create table Fine(roll_no int,Date date,Amount int);
Query OK, 0 rows affected (0.31 sec)
mysql> desc Fine;
mysql> create table Borrower(roll_no int AUTO_INCREMENT,Name
varchar(50),Date_of_issue date,Book_name varchar(50),Status varchar(10),primary
key(roll_no));
mysql> insert into
Borrower(Name,Date_of_issue,Book_name,Status)values
("Himanshu",'2023-06-15',"SEPM","Issued"),
("Abhay",'2023-08-17',"TOC","Issued"),
("Puja",'2023-06-13',"CN","Issued"),
("Geta",'2023-08-20',"TOC","Issued"),
("Kalyani",'2023-06-24',"ISEM","Issued"),
("Dhanu",'2023-07-23',"ISEM","Issued");
mysql> select* from Borrower;
mysql> delimiter $$
mysql>
create procedure studfine(roll int,nm varchar(50))
begin
declare i_date date;
declare diff int;
```

```
declare fine_amt int;
DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT"Table not found";
select Date_of_issue into i_date from Borrower where roll_no=roll and Name=nm;
select DATEDIFF(CURDATE(),i_date)into diff;
if(diff>=15 and diff<=30)
then
set fine_amt=diff*5;
insert into Fine values(roll,CURDATE(),fine_amt);
elseif(diff>30)
then
set fine_amt=diff*50;
insert into Fine values(roll,CURDATE(),fine_amt);
end if;
update Borrower set Status="Return" where roll_no=roll and Name=nm;
end $$
mysql> call studfine(1,"Pooja")
-> $$
mysql> select*from Borrower; mysql> select * from Fine;
mysql> call studfine(3,"Abhay")
-> $$
mysql> select * from Borrower;
+---------+----------+---------------+-----------+--------+
| roll_no | Name | Date_of_issue | Book_name | Status |
+---------+----------+---------------+-----------+--------+
| 1 | Pooja | 2017-06-15 | SEPM | Return |
| 2 | Himanshu | 2017-06-15 | SEPM | Issued |
| 3 | Abhay | 2017-08-17 | TOC | Return |
| 4 | Puja | 2017-06-13 | CN | Issued |
| 5 | Geta | 2017-08-25 | TOC | Issued |
| 6 | Kalyani | 2017-06-24 | ISEM | Issued |
| 7 | Dhanu | 2017-09-23 | ISEM | Issued |
+---------+----------+---------------+-----------+--------+
7 rows in set (0.00 sec)
mysql> select * from Fine;
```

```
+---------+------------+--------+
| roll_no | Date | Amount |
+---------+------------+--------+
| 1 | 2017-08-16 | 3100 |
+---------+------------+--------+
1 row in set (0.00 sec)
```

Problem Statement: -8

Develop a student database to efficiently manage and retrieve student records (Student id, Student Name, Class, address, grades, and enrolment details, subject name, attendance.

- Create Views to provide summarized insights into student performance and attendance. (Consider the attributes which shows attendance of students while creating view)
- Create Sequences to generate unique student IDs.
- Create an index on a table using student name.

Solution: -

```
create database prac8;

use prac8;


-- Create Students table with an auto-incremented student_id
CREATE TABLE Students (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    student_name VARCHAR(255) NOT NULL,
    class TEXT,
    address TEXT,
    grades TEXT,
    enrolment_details TEXT
);


-- Create Subjects table
CREATE TABLE Subjects (
    student_id INT,
    subject_name TEXT,
    attendance INT
);


-- Create an Index on the student name with a key length of 255
CREATE INDEX student_name_index
ON Students (student_name(255));
```

```
-- Create a View for summarized insights into student performance and attendance
CREATE VIEW StudentPerformance AS
SELECT s.student_id, s.student_name, AVG(sub.attendance) AS avg_attendance,
AVG(s.grades) AS avg_grades
FROM Students s
JOIN Subjects sub ON s.student_id = sub.student_id
GROUP BY s.student_id;

-- Insert sample data into Students table with Indian names
INSERT INTO Students (student_name, class, address, grades, enrolment_details)
VALUES
    ('Rahul Sharma', 'Class A', '123 Main St', 'A', '2022-01-15'),
    ('Priya Patel', 'Class B', '456 Elm St', 'B', '2022-02-20');

-- Insert sample data into Subjects table
INSERT INTO Subjects (student_id, subject_name, attendance)
VALUES
    (1, 'Math', 90),
    (2, 'Math', 85);

-- Display data from the Students table
SELECT * FROM Students;

-- Display data from the Subjects table
SELECT * FROM Subjects;
```