Assignment 1: Control Structures

1. **Write a program to Calculate Average of all numbers between n1 and n2**

Code: /*Avg.Scala*/

```
object Avg
{
	def main(args:Array[String])=
	{
		var sum=0
		var avg=0
		var cnt=0

		println("Enter n1 val")
		var n1=scala.io.StdIn.readInt()

		println("Enter n2 val")
		var n2=scala.io.StdIn.readInt()

		for(i <- n1 to n2)
		{
			sum+=i
			cnt +=1
		}
		println(s"The Sum is : $sum")
		println(s"The Count is : $cnt")

		avg=sum/cnt
		println(s"The Avg is : $avg")

	}
}
```

OutPut:

```
Enter n1 val
10
Enter n2 val
20
The Sum is : 165
The Count is : 11
The Avg is : 15
```

2. *Write A program to calculate Factorial of a number*

Code: /*Fact.Scala***/**

```
object Fact
{
	def main(args:Array[String])=
	{
		var fact=1

		println("Enter n val")
		var n=scala.io.StdIn.readInt()

		for(i <- 1 to n)
		{
			fact=fact*i
		}
		println(s"The fact of $n is : $fact")

	}
}
```

OutPut:

```
Enter n val
4
The fact of 4 is : 24
```

3. **Write a program to read five random number and check that random number are perfect or not**

Code: **/***rand.Scala***/**

```
object rand
{
	def main(args:Array[String])=
	{
		println("How Many no you want to check random")
		var n1=scala.io.StdIn.readInt()
		for(j <- 1 to n1)
		{
			var n1=scala.util.Random
			var n=n1.nextInt(10)
			var sum=0
			for(i <- 1 to n-1)
			{
				if(n%i==0)
				{
					sum=sum+i
				}
			}
			if(sum==n)
				println(s"$n is Perfect Number")
			else
				println(s"$n is not Perfect Number ")

		}
	}
}
```

OutPut:

```
How Many no you want to check random
7
3 is not Perfect Number
2 is not Perfect Number
9 is not Perfect Number
1 is not Perfect Number
6 is Perfect Number
9 is not Perfect Number
4 is not Perfect Number
```

4. Write a program to find second maximum number of four given numbers.

Code: /* SecondMax.Scala***/**

```
object SecondMax
{
	def main(args: Array[String])=
	{
		val arr = new Array[Int](4)
		println("Enter Array Element: ")
		for(i <- 0 to 3)
		{
			arr(i)=scala.io.StdIn.readInt()
		}
		scala.util.Sorting.quickSort(arr)

		println("Sorted Array Is : ")
		for(i <- 0 to (arr.length-1))
		{
			println(arr(i))
		}
		println("Second Maximum Element Is : "+arr(2))
	}
}
```

OutPut:

```
Enter Array Element:
15
45
25
35
Sorted Array Is :
15
25
35
45
Second Maximum Element Is : 35
```

5. *Write a program to find maximum and minimum of an array.*

Code: **/*** MinMax.Scala***/***

```
object MinMax{
    def main(args: Array[String])=
    {
      println("How many numbers")
      val n: Int = scala.io.StdIn.readInt()
      var m = new Array[Int](n)

      println("Enter the Array: ")
      for (i <- 0 to n - 1)
      {
        m(i) = scala.io.StdIn.readInt()
      }
      println("Array Elements are : ")
      for (i <- 0 to (m.length - 1))
      {
         println(" " + m(i))
      }

      var min:Int = m(0)
      var max:Int = m(0)

      for(i <- 1 to n-1)
      {
        if(m(i) < min)
          min = m(i)
        else if(m(i) > max)
          max = m(i)
      }
      println(s"MINIMUM = $min")
      println(s"MAXIMUM = $max")
    }
}
```

OutPut:

```
How many numbers
4
Enter the Array:
1
2
5
4
Array Elements are :
 1
 2
 5
 4
MINIMUM = 1
MAXIMUM = 5
```

6. *Write a program to calculate determinant of a matrix.*

Code: /* MatrixDet.Scala***/**

```
object Mdeter{
    def main(args: Array[String])=
    {
        var arr1=Array.ofDim[Int](10,10)
        var det=0;
        println("\n\nCalculate the determinant of a 3 x 3 matrix :")
        println("----------------------------------------------")
        println("Input elements in the first matrix ")
        for(i<-0 to 2)
        {
            for(j<-0 to 2)
            {
                arr1(i)(j)=scala.io.StdIn.readInt()
            }
        }
        println("The matrix is :\n")
        for(i<-0 to 2)
        {
            for(j<-0 to 2 )
            {
                println(arr1(i)(j)+" ");
            }
            println( )
        }
        for(i<- 0 to 2)
        {
            det = det + (arr1(0)(i)*(arr1(1)((i+1)%3)*arr1(2)((i+2)%3) -
arr1(1)((i+2)%3)*arr1(2)((i+1)%3)));
        }
        println(s"The Determinant of the matrix is: $det")
    }
}
```

OutPut:

```
Calculate the determinant of a 3 x 3 matrix :
----------------------------------------------
Input elements in the first matrix
1
12
3
4
```

```
5
6
7
8
9
The matrix is :

1
12
3

4
5
6

7
8
9

The Determinant of the matrix is: 60
```

Assignment 2: String

1. **Write a program to count uppercase letters in a string and convert it to lowercase and display the new string.**

Code: **/*** CntUpp.Scala***/***

```
object CntUpp{
    def main(args: Array[String])=
    {
       println("Enter The String: ")
       var str = scala.io.StdIn.readLine()

       var count=0

       for(e <-str)
       {
         if(e>='A' && e<='Z')
         {
            count=count+1
          }
       }

       var LStr = str.toLowerCase()
       var UStr = str.toUpperCase();

       println(s"uppercase letters:$count")

       println(s"Original String:  $str")
       println(s"String in lowercase:  $LStr")
       //println(s"String in uppercase:  $UStr")
    }
}
```

OutPut:

```
Enter The String:
Hello
uppercase letters:1
Original String:  Hello
String in lowercase:  hello
String in uppercase:  HELLO
```

2. **Write a program to read a character from user and count the number of occurrences of that character**

Code: /* Choccr.Scala***/**

```
object Choccr{
    def main(args: Array[String])=
    {
       println("Enter The String: ")
       var str = scala.io.StdIn.readLine()
       var count=0
       println("Enter the Character: ")
       var ch = scala.io.StdIn.readChar()
       for(e1 <- str)
       {      if(e1==ch)
             {
               count=count+1
             }
        }
        println(s"count of letters:$count")

    }
}
```

OutPut:

```
Enter The String:
Hello
Enter the Character:
l
count of letters:2
```

3. **Write a program to read two strings. Remove the occurrence of second string in first string.**

Code: /* RmvStr.Scala*/

```
object RmvStr{
    def main(args: Array[String])=
    {
        println("Enter The String 1 : ")
        var str1 = scala.io.StdIn.readLine()
        println("Enter The String 2 : ")
        var str2 = scala.io.StdIn.readLine()

        var str3=str1.filterNot(str2.contains)

        println(s"The String is: $str3")

    }
}
```

OutPut:

```
Enter The String 1 :
Sumaiya
Enter The String 2 :
uy
The String is: Smaia
```

4. **Create array of strings and read a string from user. Display all the elements of array containing given string.**

Code: /* disp.Scala*/

```
object disp
{
    def main(args:Array[String])=
    {
        var colors = Array("Red", "Blue", "Black", "Green")
        println("This is an array of string :")
        for(c <- colors)
        {
            println(s"${c}, ")
        }
        println("Enter pattern for matching:")
        var sbstr = scala.io.StdIn.readLine()
        var pat = sbstr.r
        println("Array elements which mathes the pattern are :")
        for(c <- colors)
        {
            var matches = pat.findFirstIn(c)
            if(matches.toString != "None")
                println(c)
        }
    }
}
```

OutPut:

```
This is an array of string :
Red,
Blue,
Black,
Green,
Enter pattern for matching:
Bl
Array elements which mathes the pattern are :
Blue
Black
```

Assignment 3: List and Set

1.Create a list of integers divisible by 3 from List containing numbers from 1 to 50.

Code: /* list.Scala*/

```
import scala.collection.mutable.ListBuffer
object div3
{
	def main(args:Array[String])=
	{
		println("Enter the Start range You want")
		val n1=scala.io.StdIn.readInt()

		println("Enter the End range You want")
		val n2=scala.io.StdIn.readInt()

		val a=new ListBuffer [Integer]()

		val x=List.range(n1,n2)
		println("Number divided by 3")
		for
		{
			i<-x
			if i%3==0
		}
		a+=i
		println(a)
	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala list.scala
Enter the Start range You want
1
Enter the End range You want
50
Number divided by 3
ListBuffer(3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48)
```

2.Create two Lists. Merge it and store the sorted in ascending order.

Code: /* merge.Scala*/

```
object merge
{
	def main(args: Array[String])=
	{
		val list1 = List(1,3,8,6)
		val list2 = List(4,2,7,5)

		println(s"list1 :  $list1")
		println(s"list2 :  $list2")

		println("Merging list1 and list2 ")
		val list3 = list1 ++ list2
		println(s"Merged list : $list3")

		val list4 = list3.sorted
		println(s"Sorted list  in Ascending Order : $list4")
	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala merge.scala
list1 :  List(1, 3, 8, 6)
list2 :  List(4, 2, 7, 5)
Merging list1 and list2
Merged list : List(1, 3, 8, 6, 4, 2, 7, 5)
Sorted list  in Ascending Order : List(1, 2, 3, 4, 5, 6, 7, 8)
```

3.Write a program to create a list of 1 to 100 numbers. Create second list from first list selecting numbers multiple of 10.

Code: /* list.Scala*/

```
import scala.collection.mutable.ListBuffer
object div3
{
	def main(args:Array[String])=
	{
		println("Enter the Start range You want")
		val n1=scala.io.StdIn.readInt()

		println("Enter the End range You want")
		val n2=scala.io.StdIn.readInt()

		val a=new ListBuffer [Integer]()

		val x=List.range(n1,n2)
		println("Number divided by 10")
		for
		{
			i<-x
			if i%10==0 && i!=0
		}
		a+=i
		println(a)
	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala list.scala
Enter the Start range You want
1
Enter the End range You want
100
Number divided by 10
ListBuffer(10, 20, 30, 40, 50, 60, 70, 80, 90)
```

4.Create a list of 50 members using function 2n+3. Create second list excluding all elements multiple of 7.

Code: /* list.Scala*/

```
import scala.collection.mutable.ListBuffer
object list
{
	def main(args:Array[String])=
	{
		println("How many Members You want to Enter: ")
		val n=scala.io.StdIn.readInt()

		val x = List.tabulate(n)(n => 2*n+3)
		println(s"List Of $n Members is: $x")

		val a=new ListBuffer [Integer]()

		val y=List.range(1,n)
		println("Number multipl by 7")
		for
		{
			i<-y
			if i%7==0
		}
		a+=i
		println(a)
	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala list.scala
How many Members You want to Enter:
50
List Of 50 Members is: List(3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29,
31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69,
71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101)
Number multipl by 7
ListBuffer(7, 14, 21, 28, 35, 42, 49)
```

5.Create a list of even numbers up to 10 and calculate its product.

Code: /* list.Scala*/

```
object list
{
	def main(args:Array[String])=
	{
		println("Enter the range You want : ")
		val n=scala.io.StdIn.readInt()

		var m=List.range(2,n+1,2)
		println(s"Even No List: $m")

		val x = m.product
		println(s"The Product of $n is : $x")

	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala list.scala
Enter the range You want :
10
Even No List: List(2, 4, 6, 8, 10)
The Product of 10 is : 3840
```

6.Write a program to create list with 10 members using function $3n^2+4n+6$

Code: /* list.Scala*/

```
object list
{
	def main(args:Array[String])=
	{
		println("How many Members You want to Enter: ")
		val n=scala.io.StdIn.readInt()

		val x = List.tabulate(n)(n => 3*n*n+4*n+6)
		println(s"List Of $n Members is: $x")

	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala list.scala
How many Members You want to Enter:
10
List Of 10 Members is: List(6, 13, 26, 45, 70, 101, 138, 181, 230, 285)
```

7.Write a program to create two sets and find common elements between them.

Code: /* set.Scala*/

```
object set
{
	def main(args:Array[String])=
	{
		val a=Set(1,9,3,7,2)
		println(s"Set 1 Element: $a")

		var b=Set(7,9,8,2,8)
		println(s"Set 2 Element: $b")

		val x = a.intersect(b)
		println(s"common elements between Sets: $x")

	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala set.scala
Set 1 Element: HashSet(1, 9, 2, 7, 3)
Set 2 Element: Set(7, 9, 8, 2)
common elements between Sets: HashSet(9, 2, 7)
```

8.Write a program to display largest and smallest element of the Set.

Code: /* set.Scala*/

```
object set
{
	def main(args:Array[String])=
	{
		val a = Set(1,9,3,7,2)
		println(s"Given Set is : $a")

		println("Max Element Given Set Is:" +a.max)

		println(s"Min Element Given Set Is:" +a.min)

	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala set.scala
Given Set is : HashSet(1, 9, 2, 7, 3)
Max Element Given Set Is:9
Min Element Given Set Is:1
```

9. Write a program to merge two sets and calculate product and average of all elements of the Set.

Code: /* set.Scala*/

```
object set
{
	def main(args:Array[String])=
	{
		val a = Set(1,2,3,4)
		println(s"Set_1: $a")

		val b = Set(5,6,7)
		println(s"Set_2: $b")

		var x = a++b
		println(s"The Merge Set Is: $x")

		val y=x.product
		println(s"The Product Of Sets is: $y")

		val count=x.size
		val sum=x.sum
		val avg=sum/count
		println(s"The Average Of given Set Is: $avg")

	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala set.scala
Set_1: Set(1, 2, 3, 4)
Set_2: Set(5, 6, 7)
The Merge Set Is: HashSet(5, 1, 6, 2, 7, 3, 4)
The Product Of Sets is: 5040
The Average Of given Set Is: 4
```

Assignment 4: Classes and Objects, MAP

1.Define a class CurrentAccount (accNo, name, balance, minBalance). Define appropriate constructors and operations withdraw(), deposit(), viewBalance(). Create an object and perform operations.

Code:
```
/* opclass.Scala*/
class CurrentAccount(var ano:Int,var nam:String,var minBal:Float)
{
    var accNo:Int=ano
    var name:String=nam
    var balance:Float=minBal
    var minBalance:Float=1000
    def withdraw()=
    {
        println("Enter the amount to be withdraw:")
        var amt:Float=scala.io.StdIn.readFloat()
        if((balance-amt)>=minBalance)
        {
            println("Balance withdraw successfully:")
            balance=balance-amt
            println(s"Remaining Balance= $balance")
        }
        else
        {
            println(s"Insufficient balance: minimum balance must be
$minBalance")
        }
    }
    def deposit()=
    {
        println(s"Balance before deposit is= $balance")
        println("Enter the amount to deposit:")
        var amt=scala.io.StdIn.readFloat()
        balance=balance+amt
        println(s"Balance after deposit= $balance")
    }
    def viewBalance()=
    {
        println(s"Account Balance= $balance")
    }
}
object CurrentAccountDemo
{
   def main(args:Array[String])=
   {
       println("Create New Account For Customer:")
       println("Enter the account number:")
       var ano=scala.io.StdIn.readInt()
       println("Enter the account holder name:")
       var nam=scala.io.StdIn.readLine()
```

```
        println("Enter the account Balance:")
        var min=scala.io.StdIn.readFloat()
        var obj=new CurrentAccount(ano,nam,min)

        while(true)
        {

println("MENU"+"\n"+"1:withdraw"+"\n"+"2:deposit"+"\n"+"3:viewBalance"+
"\n"+"4:exit")
            print("Enter the Choice:")
            var op:Int=scala.io.StdIn.readInt()
            op match
            {
                case 1 =>obj.withdraw()
                case 2 =>obj.deposit()
                case 3 =>obj.viewBalance()
                case 4 =>System.exit(0)
             }
         }
    }
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala obclass.scala
Create New Account For Customer:
Enter the account number:
125
Enter the account holder name:
sde
Enter the account Balance:
3000
MENU
1:withdraw
2:deposit
3:viewBalance
4:exit
Enter the Choice:1
Enter the amount to be withdraw:
3000
Insufficient balance: minimum balance must be 1000.0
MENU
1:withdraw
2:deposit
3:viewBalance
4:exit
Enter the Choice:2
Balance before deposit is= 3000.0
Enter the amount to deposit:
1200
Balance after deposit= 4200.0
MENU
1:withdraw
2:deposit
```

```
3:viewBalance
4:exit
Enter the Choice:3
Account Balance= 4200.0
MENU
1:withdraw
2:deposit
3:viewBalance
4:exit
Enter the Choice:4
```

2.Define a class Employee (id, name, salary). Define methods accept() and display(). Display details of employee having maximum salary.

Code:

```
/* emp.Scala*/
class Employee(var id:Int,var name:String,var salary:Double)
{
	def getSalary(): Double=
	{
		salary
	}

	def display()=
	{
		println("Employee Details are :")
		println(s"Employee ID: $id")
		println(s"Employee Name: $name")
		println(s"Employee Salary: $salary")
	}
}

object Employee
{
	def main(args: Array[String])=
	{
		println("Number of Employees")
		var n=scala.io.StdIn.readInt()
		var emp:Array[Employee]=new Array[Employee](n)
		var id:Int=0
		var name:String=null
		var salary:Double=0
		for(i<-0 until n)
		{
			println("Enter Employee Id")
			id =scala.io.StdIn.readInt()

			println("Enter Employee Name")
			name=scala.io.StdIn.readLine()

			println("Enter Employee Salary")
			salary=scala.io.StdIn.readDouble()

			println("-----------------------------")
			emp(i) =new Employee(id,name,salary)
		}
		var sal:Array[Double]=new Array[Double](n)
		for(j<-0 until n)
		{

			sal(j)=emp(j).getSalary()
			println(sal(j))
		}
```

```
            var max:Double=sal(0)
            var index:Int=0
            for(k<-0 until n-1)
            {
                  if(sal(k)<sal(k+1))
                  {
                        max=sal(k+1)
                        index=k+1
                  }
            }
            println("-----------------------------------")
            println("\nMaximum Salary ")
            emp(index).display()
      }
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala emp.scala
Number of Employees
2
Enter Employee Id
102
Enter Employee Name
sana
Enter Employee Salary
15600
-----------------------------
Enter Employee Id
106
Enter Employee Name
safa
Enter Employee Salary
45000
-----------------------------
15600.0
45000.0
-----------------------------------

Maximum Salary
Employee Details are :
Employee ID:106
Employee Name:safa
Employee Salary:45000.0
```

3.Create abstract class Order (id, description). Derive two classes PurchaseOrder and SalesOrder with members Vendor and Customer. Create object of each PurchaseOrder and SalesOrder. Display the details of each account

Code:

```
/* abs.Scala*/
abstract class Order(id:Int,des:String)
{
	var Id:Int=id
	var Des:String=des
}

class PurchaseOrder(vendor:String,customer:String,id:Int,des:String) extends
Order(id,des)
{
	var vend:String=vendor
	var cust:String=customer
	var Id_1:Int=id
	var Des_1:String=des

	def PurchOrder()=
	{
		print(s"$Id_1 \t $Des_1 \t $vend \t $cust \n")
	}
}

class SalesOrder(vendor:String,customer:String,id:Int,des:String) extends
Order(id,des)
{
	var vend:String=vendor
	var cust:String=customer
	var Id_2:Int=id
	var Des_2:String=des

	def SaleOrder()=
	{
		print(s"$Id_2 \t $Des_2 \t $vend \t $cust \n")
	}
}

object Orders
{
	def main(a:Array[String])=
	{
		val n1=new SalesOrder("Himmat","Aniket",1,"sold")
		val n2=new PurchaseOrder("Sunny","Sagar",2,"Purchased")
		while(true)
		{

println("MENU"+"\n"+"1:SalesOrder"+"\n"+"2:PurchaseOrder"+"\n"+"3:Exit")
			print("Enter the Choice:")
			var op:Int=scala.io.StdIn.readInt()
```

```
                    op match{
                        case 1=>n1.SaleOrder()

                        case 2=>n2.PurchOrder()

                        case 3=>System.exit(0)
                    }
                }
            }
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala abs.scala
MENU
1:SalesOrder
2:PurchaseOrder
3:Exit
Enter the Choice:1
1       sold    Himmat          Aniket
MENU
1:SalesOrder
2:PurchaseOrder
3:Exit
Enter the Choice:2
2       Purchased       Sunny   Sagar
MENU
1:SalesOrder
2:PurchaseOrder
3:Exit
Enter the Choice:3
```

4.Write a program to create map with Rollno and FirstName. Print all student information with same FirstName.

Code: /* map.Scala*/

```
object map
{
	def main(a:Array[String])=
	{
		var map=Map(1->"mayuri",2->"mayuri",3->"Chinmay",4-
>"mayuri",5->"mayi")
		for((k1,v1) <- map)
		{
			for((k2,v2)<-map)
			{
				if(v1==v2 && k1!=k2)
				{
					println(s" Roll No: $k1  FirstName: $v1")
				}
			}
		}
	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala map.scala
 Roll No: 1  FirstName: mayuri
 Roll No: 1  FirstName: mayuri
 Roll No: 2  FirstName: mayuri
 Roll No: 2  FirstName: mayuri
 Roll No: 4  FirstName: mayuri
 Roll No: 4  FirstName: mayuri
```

5.Write a user defined functions to convert lowercase letter to uppercase and call the function using Map.

Code: /* callmap.Scala*/

```
object callmap
{
	def main(arg: Array[String])=
	{
		var a = scala.io.StdIn.readLine("Enter the letter:\n")
		con(a)
	}

	def con(a : String)=
	{
		var cap = a.map(c => c.toUpper)
		println("In Upparcase")
		println(cap)
	}
}
```

OutPut:

```
C:\scala3-3.2.1\bin>scala callmap.scala
Enter the letter:
sana
In Upparcase
SANA
```

Solve the following database using MongoDB :

1. **Create a database with name 'Company'.**

An 'Employee' is a collection of documents with the following Fields :

a. Employee ID
b. First Name
c. Last Name
d. Email
e. PhoneNo.
f. Address
h. Designation
i. Experience
j. Date of Joining
k. Birthdate

A 'Transaction' is a collection of documents with the following fields:

a. Transaction Id,
b. Transaction Date
c. Name
d. Transaction Details
e. Payment
f. Remark

Solve the Following Queries:

1. Update designation of an employee having Employee id as_____.
2. Update the designation of an employee named "_____ " from supervisor to manager
3. Delete the transaction made by "_____" employee on the given date.
4. Change the address of an employee having Employee id as _____.
5. Sort the employees in the descending order of their designation.
6. Count the total number of employees in a collection.
7. Change the address of an employee having Employee id as _____
8. Display all the documents of both the collections in a formatted manner.
9. Update designation of an employee having Employee id as_____.
10. Update salary of all employees by giving an increment of Rs.4000.

11. Delete the transaction made by "_____" employee on the given date.
12. Change the address of an employee having Employee id as _____.

```
> use Company3
switched to db Company3

Employee:
> db.createCollection('Employee')
{ "ok" : 1 }
> db.Employee.createIndex({"eid":1},{unique:true})
{
	"createdCollectionAutomatically" : false,
	"numIndexesBefore" : 1,
	"numIndexesAfter" : 2,
	"ok" : 1
}
>db.Employee.insertOne({eid:101,fname:"Aasha",lname:"Roy",email:"aasha.roy@gmail.com",phoneno:1254678,address:"Pune",salary:20000,desg:"Admin",Exp:3,doj:"2020-02-12",dob:"1998-08-24"})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381a96db6acb427e3f123c0")
}
>db.Employee.insertOne({eid:102,fname:"Pritee",lname:"Roy","Email":"Pritee.123@gmail.com",Phone:9822151654 ,addrs:"Bombay", salary:18000,desg:"Supervisor",exp:2,doj:"2020-07-01",dob:"1996-08-21"})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381a9a1b6acb427e3f123c1")
}
>db.Employee.insertOne({eid:103,fname:"Mina",lname:"Sharma","Email":"mina.sharma@gmail.com",Phone:9011568975,addrs:"Pune",salary:21500,desg:"Clerk",exp:7,doj:"2017-06-01",dob:"1987-03-22"})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381a9afb6acb427e3f123c2")
}
>db.Employee.insertOne({eid:104,fname:"Geeta",lname:"Patil","Email":"geeta_patil@yahoo.com",Phone:7856452598,addrs:"Pune",salary:23000,desg:"Admin",exp:8,doj:"2018-06-15",dob:"1990-02-28"})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381aa3ab6acb427e3f123c3")
}
```

Transaction:

```
> db.createCollection("Transaction")
{ "ok" : 1 }
> db.Transaction.createIndex({"tid":1},{unique:true})
{
	"createdCollectionAutomatically" : false,
	"numIndexesBefore" : 1,
	"numIndexesAfter" : 2,
	"ok" : 1
}

> db.transaction.insertOne({tid:1,tdate:"2021-05-21",name:"T1",tdetails:"Grocery",payment:2500,remark:"done",eid:101})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381accdb6acb427e3f123c4")
}
> db.transaction.insertOne({tid:2,tdate:"2021-05-21",name:"T2",tdetails:"Snacks",payment:2280,remark:"incomplete",eid:102})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381acf0b6acb427e3f123c5")
}
> db.transaction.insertOne({tid:3,tdate:"2021-06-25",name:"T3",tdetails:"Sweets",payment:2690,remark:"complete",eid:103})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381ad6eb6acb427e3f123c6")
}
> db.transaction.insertOne({tid:4,tdate:"2021-07-23",name:"T4",tdetails:"Sweets",payment:2700,remark:"complete",eid:104})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381ada1b6acb427e3f123c7")
}
> db.transaction.insertOne({tid:5,tdate:"2021-08-25",name:"T4",tdetails:"Sweets",payment:2700,remark:"complete",eid:104})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("6381adb8b6acb427e3f123c8")
}
```

```
> db.transaction.find().pretty()
{
	"_id" : ObjectId("6381accdb6acb427e3f123c4"),
	"tid" : 1,
	"tdate" : "2021-05-21",
	"name" : "T1",
	"tdetails" : "Grocery",
	"payment" : 2500,
	"remark" : "done",
	"eid" : 101
}
{
	"_id" : ObjectId("6381acf0b6acb427e3f123c5"),
	"tid" : 2,
	"tdate" : "2021-05-21",
	"name" : "T2",
	"tdetails" : "Snacks",
	"payment" : 2280,
	"remark" : "incomplete",
	"eid" : 102
}
{
	"_id" : ObjectId("6381ad6eb6acb427e3f123c6"),
	"tid" : 3,
	"tdate" : "2021-06-25",
	"name" : "T3",
	"tdetails" : "Sweets",
	"payment" : 2690,
	"remark" : "complete",
	"eid" : 103
}
{
	"_id" : ObjectId("6381ada1b6acb427e3f123c7"),
	"tid" : 4,
	"tdate" : "2021-07-23",
	"name" : "T4",
	"tdetails" : "Sweets",
	"payment" : 2700,
	"remark" : "complete",
	"eid" : 104
}
{
	"_id" : ObjectId("6381adb8b6acb427e3f123c8"),
	"tid" : 5,
	"tdate" : "2021-08-25",
	"name" : "T4",
	"tdetails" : "Sweets",
	"payment" : 2700,
	"remark" : "complete",
	"eid" : 104
}
```

Queries:

1) **Update designation of an employee having Employee id as 101**

```
> db.Employee.updateOne({eid:101},{$set:{"desg":"Staff"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.Employee.find({eid:101}).pretty()
{
        "_id" : ObjectId("638ad9fdd760bfac5aa08924"),
        "eid" : 101,
        "fname" : "Aasha",
        "lname" : "Roy",
        "email" : "aasha.roy@gmail.com",
        "phoneno" : 1254678,
        "address" : "Pune",
        "salary" : 20000,
        "desg" : "Staff",
        "Exp" : 3,
        "doj" : "2020-02-12",
        "dob" : "1998-08-24",
        "addrs" : "Mumbai"
}
```

2) **Update the designation of an employee named “Pritee” from supervisor to manager**

```
> db.Employee.updateOne({fname:"Pritee"},{$set:{"desg":"Manager"}})
> db.Employee.find({eid:102}).pretty()
{
        "_id" : ObjectId("638ada09d760bfac5aa08925"),
        "eid" : 102,
        "fname" : "Pritee",
        "lname" : "Roy",
        "Email" : "Pritee.123@gmail.com",
        "Phone" : 9822151654,
        "addrs" : "Bombay",
        "salary" : 18000,
        "desg" : "Manager",
        "exp" : 2,
        "doj" : "2020-07-01",
        "dob" : "1996-08-21"
}
```

3) Delete the transaction made by "Aasha" employee on the given date : 2021-08-25

> db.transaction.deleteOne({eid:103,tdate:"2021-06-25"})

```
{ "acknowledged" : true, "deletedCount" : 1 }
```

> db.transaction.find().pretty()

```
{
	"_id" : ObjectId("638ada68d760bfac5aa08928"),
	"tid" : 1,
	"tdate" : "2021-05-21",
	"name" : "T1",
	"tdetails" : "Grocery",
	"payment" : 2500,
	"remark" : "done",
	"eid" : 101
}
{
	"_id" : ObjectId("638ada72d760bfac5aa08929"),
	"tid" : 2,
	"tdate" : "2021-05-21",
	"name" : "T2",
	"tdetails" : "Snacks",
	"payment" : 2280,
	"remark" : "incomplete",
	"eid" : 102
}
{
	"_id" : ObjectId("638ada8fd760bfac5aa0892b"),
	"tid" : 4,
	"tdate" : "2021-07-23",
	"name" : "T4",
	"tdetails" : "Sweets",
	"payment" : 2700,
	"remark" : "complete",
	"eid" : 104
}
{
	"_id" : ObjectId("638ada9ed760bfac5aa0892c"),
	"tid" : 5,
	"tdate" : "2021-08-25",
	"name" : "T4",
	"tdetails" : "Sweets",
	"payment" : 2700,
	"remark" : "complete",
	"eid" : 104
}
```

4) Change the address of an employee having employee id as 101

> db.Employee.update({eid:101},{$set:{addrs:'Banglore'}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.Employee.find().pretty()


```
{
	"_id" : ObjectId("638ad9fdd760bfac5aa08924"),
	"eid" : 101,
	"fname" : "Aasha",
	"lname" : "Roy",
	"email" : "aasha.roy@gmail.com",
	"phoneno" : 1254678,
	"address" : "Pune",
	"salary" : 20000,
	"desg" : "Staff",
	"Exp" : 3,
	"doj" : "2020-02-12",
	"dob" : "1998-08-24",
	"addrs" : "Banglore"
}
{
	"_id" : ObjectId("638ada09d760bfac5aa08925"),
	"eid" : 102,
	"fname" : "Pritee",
	"lname" : "Roy",
	"Email" : "Pritee.123@gmail.com",
	"Phone" : 9822151654,
	"addrs" : "Bombay",
	"salary" : 18000,
	"desg" : "Manager",
	"exp" : 2,
	"doj" : "2020-07-01",
	"dob" : "1996-08-21"
}
{
	"_id" : ObjectId("638ada1bd760bfac5aa08926"),
	"eid" : 103,
	"fname" : "Mina",
	"lname" : "Sharma",
	"Email" : "mina.sharma@gmail.com",
	"Phone" : 9011568975,
	"addrs" : "Pune",
	"salary" : 21500,
	"desg" : "Clerk",
	"exp" : 7,
	"doj" : "2017-06-01",
	"dob" : "1987-03-22"
}
{
	"_id" : ObjectId("638ada26d760bfac5aa08927"),
	"eid" : 104,
	"fname" : "Geeta",
```

```
    "lname" : "Patil",
    "Email" : "geeta_patil@yahoo.com",
    "Phone" : 7856452598,
    "addrs" : "Pune",
    "salary" : 23000,
    "desg" : "Admin",
    "exp" : 8,
    "doj" : "2018-06-15",
    "dob" : "1990-02-28"
}
```

5) **Sort the employees in the descending order of their designation**

> db.Employee.find().sort({desg:-1}).pretty()

```
{
        "_id" : ObjectId("638ad9fdd760bfac5aa08924"),
        "eid" : 101,
        "fname" : "Aasha",
        "lname" : "Roy",
        "email" : "aasha.roy@gmail.com",
        "phoneno" : 1254678,
        "address" : "Pune",
        "salary" : 20000,
        "desg" : "Staff",
        "Exp" : 3,
        "doj" : "2020-02-12",
        "dob" : "1998-08-24",
        "addrs" : "Banglore"
}
{
        "_id" : ObjectId("638ada09d760bfac5aa08925"),
        "eid" : 102,
        "fname" : "Pritee",
        "lname" : "Roy",
        "Email" : "Pritee.123@gmail.com",
        "Phone" : 9822151654,
        "addrs" : "Bombay",
        "salary" : 18000,
        "desg" : "Manager",
        "exp" : 2,
        "doj" : "2020-07-01",
        "dob" : "1996-08-21"
}
{
        "_id" : ObjectId("638ada1bd760bfac5aa08926"),
        "eid" : 103,
        "fname" : "Mina",
        "lname" : "Sharma",
        "Email" : "mina.sharma@gmail.com",
        "Phone" : 9011568975,
        "addrs" : "Pune",
        "salary" : 21500,
```

```
        "desg" : "Clerk",
        "exp" : 7,
        "doj" : "2017-06-01",
        "dob" : "1987-03-22"
    }
    {
        "_id" : ObjectId("638ada26d760bfac5aa08927"),
        "eid" : 104,
        "fname" : "Geeta",
        "lname" : "Patil",
        "Email" : "geeta_patil@yahoo.com",
        "Phone" : 7856452598,
        "addrs" : "Pune",
        "salary" : 23000,
        "desg" : "Admin",
        "exp" : 8,
        "doj" : "2018-06-15",
        "dob" : "1990-02-28"
    }
```

6) Count the total number of employees in a collection

> db.Employee.find().count()

4

7) Display all the documents of both the collections in a formatted manner.

>db.Employee.find().pretty()

```
{
    "_id" : ObjectId("638ad9fdd760bfac5aa08924"),
    "eid" : 101,
    "fname" : "Aasha",
    "lname" : "Roy",
    "email" : "aasha.roy@gmail.com",
    "phoneno" : 1254678,
    "address" : "Pune",
    "salary" : 20000,
    "desg" : "Staff",
    "Exp" : 3,
    "doj" : "2020-02-12",
    "dob" : "1998-08-24",
    "addrs" : "Banglore"
}
{
    "_id" : ObjectId("638ada09d760bfac5aa08925"),
    "eid" : 102,
    "fname" : "Pritee",
    "lname" : "Roy",
    "Email" : "Pritee.123@gmail.com",
    "Phone" : 9822151654,
```

```
        "addrs" : "Bombay",
        "salary" : 18000,
        "desg" : "Manager",
        "exp" : 2,
        "doj" : "2020-07-01",
        "dob" : "1996-08-21"
}
{
        "_id" : ObjectId("638ada1bd760bfac5aa08926"),
        "eid" : 103,
        "fname" : "Mina",
        "lname" : "Sharma",
        "Email" : "mina.sharma@gmail.com",
        "Phone" : 9011568975,
        "addrs" : "Pune",
        "salary" : 21500,
        "desg" : "Clerk",
        "exp" : 7,
        "doj" : "2017-06-01",
        "dob" : "1987-03-22"
}
{
        "_id" : ObjectId("638ada26d760bfac5aa08927"),
        "eid" : 104,
        "fname" : "Geeta",
        "lname" : "Patil",
        "Email" : "geeta_patil@yahoo.com",
        "Phone" : 7856452598,
        "addrs" : "Pune",
        "salary" : 23000,
        "desg" : "Admin",
        "exp" : 8,
        "doj" : "2018-06-15",
        "dob" : "1990-02-28"
}
> db.transaction.find().pretty()
{
        "_id" : ObjectId("638ada68d760bfac5aa08928"),
        "tid" : 1,
        "tdate" : "2021-05-21",
        "name" : "T1",
        "tdetails" : "Grocery",
        "payment" : 2500,
        "remark" : "done",
        "eid" : 101
}
```

```
{
	"_id" : ObjectId("638ada72d760bfac5aa08929"),
	"tid" : 2,
	"tdate" : "2021-05-21",
	"name" : "T2",
	"tdetails" : "Snacks",
	"payment" : 2280,
	"remark" : "incomplete",
	"eid" : 102
}
{
	"_id" : ObjectId("638ada8fd760bfac5aa0892b"),
	"tid" : 4,
	"tdate" : "2021-07-23",
	"name" : "T4",
	"tdetails" : "Sweets",
	"payment" : 2700,
	"remark" : "complete",
	"eid" : 104
}
{
	"_id" : ObjectId("638ada9ed760bfac5aa0892c"),
	"tid" : 5,
	"tdate" : "2021-08-25",
	"name" : "T4",
	"tdetails" : "Sweets",
	"payment" : 2700,
	"remark" : "complete",
	"eid" : 104
}
```


8) Update salary of all employees by giving an increment of Rs. 4000
(To display only salary column) :

>db.Employee.updateMany({},{$inc:{salary:4000}})


```
{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 4 }
```


>db.Employee.find().pretty()


```
{
	"_id" : ObjectId("638ad9fdd760bfac5aa08924"),
	"eid" : 101,
	"fname" : "Aasha",
	"lname" : "Roy",
	"email" : "aasha.roy@gmail.com",
	"phoneno" : 1254678,
	"address" : "Pune",
	"salary" : 28000,
	"desg" : "Staff",
	"Exp" : 3,
```

```
	"doj" : "2020-02-12",
	"dob" : "1998-08-24",
	"addrs" : "Banglore"
}
{
	"_id" : ObjectId("638ada09d760bfac5aa08925"),
	"eid" : 102,
	"fname" : "Pritee",
	"lname" : "Roy",
	"Email" : "Pritee.123@gmail.com",
	"Phone" : 9822151654,
	"addrs" : "Bombay",
	"salary" : 26000,
	"desg" : "Manager",
	"exp" : 2,
	"doj" : "2020-07-01",
	"dob" : "1996-08-21"
}
{
	"_id" : ObjectId("638ada1bd760bfac5aa08926"),
	"eid" : 103,
	"fname" : "Mina",
	"lname" : "Sharma",
	"Email" : "mina.sharma@gmail.com",
	"Phone" : 9011568975,
	"addrs" : "Pune",
	"salary" : 29500,
	"desg" : "Clerk",
	"exp" : 7,
	"doj" : "2017-06-01",
	"dob" : "1987-03-22"
}
{
	"_id" : ObjectId("638ada26d760bfac5aa08927"),
	"eid" : 104,
	"fname" : "Geeta",
	"lname" : "Patil",
	"Email" : "geeta_patil@yahoo.com",
	"Phone" : 7856452598,
	"addrs" : "Pune",
	"salary" : 31000,
	"desg" : "Admin",
	"exp" : 8,
	"doj" : "2018-06-15",
	"dob" : "1990-02-28"
}
```

2. Create a database with the name 'Movie'.

A 'Film' is a collection of documents with the following fields:

a. Film Id

b. Title of the film

c. Year of release

d. Category / Genre (like adventure, action, sci-fi, romantic etc.) A film can belong to more than genre

e. Actors (A Film can have more than one actor)

f. Director (A Film can have more than one director)

g. Release details (Place,date,rating)

An 'Actor' is a collection of documents with the following fields:

a. Actor Id

b. Name

d. Address

e. Contact Details

f. Age of an actor.

Solve the Following Queries:

1. Delete an actor named "Salman Khan ".
2. Delete all actors from an 'Actor' collection who have age greater than " 25 "
3. Update the actor's address where Actor Id is "A102".
4. Insert at least one document with film belonging to two genres
5. Insert at least one document with film belonging to three genres.
6. Insert at least one document with film that is released at more than one place and on two different dates.
7. Insert at least three documents with the films released in the same year.
8. Delete an actor named "Mr. Khan ".
9. Find the titles of all the films starting with the letter 'R' released during the year 2009
10. Find the list of films acted by an actor " ".
11. Insert at least one document with film belonging to two categories.
12. Delete all actors from an 'Actor' collection who have age greater than "_____".

```
> use movie
switched to db movie
```

Collection Movie

```
> db.createCollection('Film')
{ "ok" : 1 }
> db.Employee.createIndex({Film_id:1},{unique:true})
```

```
    {
        "createdCollectionAutomatically" : true,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
    }
```

Collection Actor

> **db.createCollection('Actor')**
{ "ok" : 1 }
> **db.Actor.createIndex({'Actor_id':1},{unique:true})**
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}

>**db.Actor.insert({Actor_id:'A101',ActorName:"Akshay",Address:"Mumbai",ContactNo:"9256897540",Age:23})**
WriteResult({ "nInserted" : 1 })

> **db.Actor.insert({Actor_id:'A102',ActorName:"Salman Khan",Address:"Pune",ContactNo:"9568975410",Age:55})**
WriteResult({ "nInserted" : 1 })

> db.Actor.insert({Actor_id:"A103",ActorName:"Amir Khan",Address:"Mumbai",ContactNo:"9164975450",Age:45})
WriteResult({ "nInserted" : 1 })

Queries

1. **Insert at least one document with film belonging to two genres**

OR

2. **Insert at least one document with film belonging to two categories.**

Code:

```
> db.Film.insert({Film_id:1, Title:'Super Heros',Year_of_Release:2020,Genre:['Action','Drama'],ActorsName:['Salman Khan','Rocky Roy'],DiretorName:['Sanjay','Mr. Khan'],ReleaseDetails:[{Place:'Pune',Date:'04/15/2020',Rating:'A'}]})
```

OutPut:

```
WriteResult({ "nInserted" : 1 })
```

3. **Insert at least one document with film that is released at more than one place and on two different dates.**

Code:

```
>db.Film.insert({Film_id:2, Title:'ABCD',Year_of_Release:2020,Genre:['Action','Drama','Comedy'],ActorsName:['Sanjay Dutt','Mahima'],DiretorName:['Mahesh'],ReleaseDetails:[{Place:['Pune','Mumbai'],Date:['04/15/2020','04/16/2020'],Rating:'A'}]})
```

OutPut:

```
WriteResult({ "nInserted" : 1 })
```

4. **Insert at least three documents with the films released in the same year.**

Code:

```
db.Film.insert({Film_id:3, Title:'PQRS',Year_of_Release:2020,Genre:['Comedy'],ActorsName:['Salim','Minal'],DiretorName:['Mahesh'],ReleaseDetails:[{Place:'Mumbai',Date:'05/18/2020',Rating:'A+'}]})
```

OutPut:

```
WriteResult({ "nInserted" : 1 })
```

Displaying Data from Film

```
> db.Film.find().pretty()
{
        "_id" : ObjectId("638aef1adb70366965194340"),
        "Film_id" : 1,
        "Title" : "Super Heros",
```

```
    "Year_of_Release" : 2020,
    "Genre" : [
        "Action",
        "Drama"
    ],
    "ActorsName" : [
        "Salman Khan",
        "Rocky Roy"
    ],
    "DiretorName" : [
        "Sanjay",
        "Mr. Khan"
    ],
    "ReleaseDetails" : [
        {
            "Place" : "Pune",
            "Date" : "04/15/2020",
            "Rating" : "A"
        }
    ]
}
{
    "_id" : ObjectId("638aefaddb70366965194341"),
    "Film_id" : 2,
    "Title" : "ABCD",
    "Year_of_Release" : 2020,
    "Genre" : [
        "Action",
        "Drama",
        "Comedy"
    ],
    "ActorsName" : [
        "Sanjay Dutt",
        "Mahima"
    ],
    "DiretorName" : [
        "Mahesh"
    ],
    "ReleaseDetails" : [
        {
            "Place" : [
                "Pune",
                "Mumbai"
```

```
                ],
                "Date" : [
                        "04/15/2020",
                        "04/16/2020"
                ],
                "Rating" : "A+"
        }
    ]
}
{
    "_id" : ObjectId("638aefc7db70366965194342"),
    "Film_id" : 3,
    "Title" : "PQRS",
    "Year_of_Release" : 2020,
    "Genre" : [
        "Comedy"
    ],
    "ActorsName" : [
        "Salim",
        "Minal"
    ],
    "DiretorName" : [
        "Mahesh"
    ],
    "ReleaseDetails" : [
        {
            "Place" : "Mumbai",
            "Date" : "05/18/2020",
            "Rating" : "A+"
        }
    ]
}
```

5. Update the actor's address where Actor Id is "A102".

For Checking

> **db.Actor.find().pretty()**

```
{
    "_id" : ObjectId("638af200db70366965194343"),
    "Actor_id" : "A101",
    "ActorName" : "Akshay",
    "Address" : "Mumbai",
    "ContactNo" : "9256897540",
    "Age" : 23
```

```
}
{
	"_id" : ObjectId("638af237db70366965194344"),
	"Actor_id" : "A102",
	"ActorName" : "Salman Khan",
	"Address" : "Pune", //update pending
	"ContactNo" : "9568975410",
	"Age" : 55
}
{
	"_id" : ObjectId("638af245db70366965194345"),
	"Actor_id" : "A103",
	"ActorName" : "Amir Khan",
	"Address" : "Mumbai",
	"ContactNo" : "9164975450",
	"Age" : 45
}
```

Code:

```
> db.Actor.update({Actor_id:'A102'},{$set:{'Address':"Mumbai"}})
```

OutPut:

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

For Checking

```
> db.Actor.find({Actor_id:'A102'}).pretty()
{
	"_id" : ObjectId("637f1a78eee8caa59ac0f968"),
	"Actor_id" : "A102",
	"ActorName" : "Salman Khan",
	"Address" : "Mumbai",
	"ContactNo" : "9568975410",
	"Age" : 55
}
```

6. **Find the titles of all the films starting with the letter 'A' released during the year 2020**

Code:

```
>db.Film.find({$and:[{Title:{$regex:/^A/}},{Year_of_Release:2020}]}).
pretty()
```

Output:

```
{
	"_id" : ObjectId("638aefaddb70366965194341"),
	"Film_id" : 2,
	"Title" : "ABCD",
	"Year_of_Release" : 2020,
	"Genre" : [
```

```
            "Action",
            "Drama",
            "Comedy"
      ],
      "ActorsName" : [
            "Sanjay Dutt",
            "Mahima"
      ],
      "DiretorName" : [
            "Mahesh"
      ],
      "ReleaseDetails" : [
            {
                  "Place" : [
                        "Pune",
                        "Mumbai"
                  ],
                  "Date" : [
                        "04/15/2020",
                        "04/16/2020"
                  ],
                  "Rating" : "A+"
            }
      ]
}
```

7. **Find the list of films acted by an actor "Salman Khan".**

Code:

```
>db.Film.find({ActorsName:'Salman Khan'}).pretty()
```

Output:

```
{
        "_id" : ObjectId("63a3068420d3755ae40a9aaf"),
        "Film_id" : 1,
        "Title" : "Super Heros",
        "Year_of_Release" : 2020,
        "Genre" : [
                "Action",
                "Drama"
        ],
        "ActorsName" : [
                "Salman Khan",
                "Rocky Roy"
        ],
        "DiretorName" : [
                "Sanjay",
                "Mr. Khan"
        ],
```

```
        "ReleaseDetails" : [
                {
                        "Place" : "Pune",
                        "Date" : "04/15/2020",
                        "Rating" : "A"
                }
        ]
}
```

8. **Insert at least one document with film belonging to three genres.**

Code:

```
>db.Film.insert({Film_id:2,
Title:'ABCD',Year_of_Release:2020,Genre:['Action','Drama','Comedy'],
ActorsName:['Sanjay
Dutt','Mahima'],DiretorName:['Mahesh'],ReleaseDetails:[{Place:['Pune','M
umbai'],Date:['04/15/2020','04/16/2020'],Rating:'A'}]})
```

OutPut:

```
WriteResult({ "nInserted" : 1 })
```

9. **Delete an actor named "Salman Khan ".**

Code:

```
> db.Actor.remove({ActorName:'Salman Khan'})
```

OutPut:

```
WriteResult({ "nRemoved" : 1 })
```

10. **Delete all actors from an 'Actor' collection who have age greater than "45".**

OR

11. **Delete all actors from an 'Actor' collection who have age greater than "____".**

Code:

```
> db.Actor.remove({Age:{$gt:45}})
```

OutPut:

```
WriteResult({ "nRemoved" : 0 })
```

Neo4j

Solve the following database using Neo4j.

1. **Consider a Song database, with labels as Artists, Song, Recording_company, Recoding_studio, song author etc.**

Relationships can be as follows:

Artist → [Performs] → Song → [Written by] → Song_author.

Song → [Recorded in] → Recording Studio →[managed by] → Recording_ Company

Recording Company → [Finances] → Song

Solve the Following Queries:

1. List the names of artist performing the song "...."
2. Name the songs recorded by the studio "....."
3. List the names of songs written by " ....."
4. List the names of record companies who have financed for the song"...."
5. List the names of artist performing the song "....."
6. List the names of songs written by "....."
7. List the names of artist performing the song"....."
8. List the names of record companies who have financed for the song"...."
9. List the names of artist performing the song "....."
10. List the names of songs written by "....."
11. List the names of artist performing the song"....."
12. List the names of record companies who have financed for the song"...."

Creating Label Artists

```
create(Artist1:a1{name:'Shreya Ghoshal',age:25,addr:'Pune'}) return(Artist1)

create(Artist2:a2{name:'Aasha Bhosle',age:55,addr:'Mumbai'}) return(Artist2)

create(Artist3:a3{name:'Abhijit Sawant',age:35,addr:'Nagar'}) return(Artist3)

create(Artist4:a4{name:'Aarya Ambekar',age:24,addr:'Nagpur'}) return(Artist4)
```

Creating Label Song

```
create(Song1:s1{name:'Song1',lyrics:'Abcd'}) return(Song1)

create(Song2:s2{name:'Song2',lyrics:'XyZ'}) return(Song2)

create(Song3:s3{name:'Song3',lyrics:'pqrs'}) return(Song3)

create(Song4:s4{name:'Song4',lyrics:'tuvw'}) return(Song4)
```

Creating Label Recording_Company

```
create(Recording_Company1:rc1{name:'T-Series',Est:2010}) return(Recording_Company1)

create(Recording_Company2:rc2{name:'HMV',Est:2011}) return(Recording_Company2)

create(Recording_Company3:rc3{name:'Sangeet',Est:2012}) return(Recording_Company3)

create(Recording_Company4:rc4{name:'Sureel',Est:2013}) return(Recording_Company4)
```

Creating Label Recording_Studio

```
create(Recording_Studio1:rs1{name:'R.K',location:'Pune'}) return(Recording_Studio1)

create(Recording_Studio2:rs2{name:'R.J',location:'Delhi'}) return(Recording_Studio2)

create(Recording_Studio3:rs3{name:'Saa',location:'Pune'}) return(Recording_Studio3)

create(Recording_Studio4:rs4{name:'Sunshine',location:'Mumbai'}) return(Recording_Stu
dio4)
```

Creating Label Song_Author

```
create(Song_Author1:sa1{name:'Nitin',addr:'Pune'}) return(Song_Author1)

create(Song_Author2:sa2{name:'Seema',addr:'Mumbai'}) return(Song_Author2)

create(Song_Author3:sa3{name:'Resham',addr:'Mumbai'}) return(Song_Author3)

create(Song_Author4:sa4{name:'Sameer',addr:'Nagpur'}) return(Song_Author4)
```

Creating Relationships:

Artist →[Preforms]→Song→[Written By]→Song_Author

Song→[Recorded in]→Recording Studio→[Managed by]→Recording_Company

Recording_Company→[Finances]→Song

Note:

To Delete Node With all Its RelationShips

MATCH(s1{name:'Song1'}) DETACH DELETE s1

Artist -- [Preforms]→Song -- [Written By]→Song_Author

```
Match(a:a1{name:'Shreya Ghoshal'}),(b:s1{name:'Song1'}),(c:sa1{name:'Nitin'})create(a)-[r1:Performs]->(b)-[r2:Written_By]->(c) return r1,r2

Match(a:a2{name:'Aasha Bhosle'}),(b:s2{name:'Song2'}),(c:sa2{name:'Seema'})create(a)-[r3:Performs]->(b)-[r4:Written_By]->(c) return r3,r4

Match(a:a2{name:'Aasha Bhosle'}),(b:s2{name:'Song2'}),(c:sa3{name:'Resham'})create(a)-[r5:Performs]->(b)-[r6:Written_By]->(c) return r5,r6
```

Song→[Recorded in]→Recording Studio→[Managed by]→Recording_Company

```
Match(a:s1{name:'Song1'}),(b:rs1{name:'R.K'}),(c:rc1{name:'T-Series'})create(a)-[r1:Recorded_in]->(b)-[r2:Managed_By]->(c) return r1,r2

Match(a:s2{name:'Song2'}),(b:rs1{name:'R.K'}),(c:rc2{name:'HMV'})create(a)-[r1:Recorded_in]->(b)-[r2:Managed_By]->(c) return r1,r2

 Match(a:s3{name:'Song3'}),(b:rs2{name:'R.J'}),(c:rc3{name:'Sangeet'})create(a)-[r1:Recorded_in]->(b)-[r2:Managed_By]->(c) return r1,r2
```

Recording_Company→[Finances]→Song

```
Match(a:rc1{name:'T-Series'}),(b:s1{name:'Song1'})create(a)-[r1:Finances]->(b) return r1

Match(a:rc1{name:'T-Series'}),(b:s2{name:'Song2'})create(a)-[r1:Finances]->(b) return r1

Match(a:rc2{name:'HMV'}),(b:s3{name:'Song3'})create(a)-[r1:Finances]->(b) return r1

Match(a:rc4{name:'Sureel'}),(b:s4{name:'Song4'})create(a)-[r1:Finances]->(b) return r1
```

Queries:

1. **List the names Of artist Performing The Song “Song1”**

Query:

```
Match(a)-[:Performs]->(:s1{name:'Song1'})return a.name
```

Output:

```
song$ Match(a)-[:Performs]→(:s1{name:'Song1'})return a.name
```

	a.name
1	"Shreya Ghoshal"

2. **Name The Songs Recorded by the Studio “R.K”**

Query:

```
Match(a)-[:Recorded_in]->(b:rs1{name:'R.K'})return a.name
```

Output:

```
song$ Match(a)-[:Recorded_in]→(b:rs1{name:'R.K'})return a.name
```

	a.name
1	"Song2"
2	"Song1"

3. List The name Song Written By “Nitin”

Query:

```
Match(a)-[:Written_By]->(b:sa1{name:'Nitin'})return a.name
```

Output:

```
song$ Match(a)-[:Written_By]→(b:sa1{name:'Nitin'})return a.name
```

	a.name
1	"Song1"

4. List the Names of record Companies Who have Financed for the Song”Song1”

Query:

```
Match(a)-[:Finances]->(:s1{name:'Song1'})return a.name
```

Output:

```
song$ Match(a)-[:Finances]→(:s1{name:'Song1'})return a.name
```

	a.name
1	"T-Series"

2. **Solve the following database using Neo4j.**

Consider an Employee database, with a minimal set of labels as follows
Employee: denotes a person as an employee of the organization
Department: denotes the different departments, in which employees work.
Skillset: A list of skills acquired by an employee
Projects: A list of projects in which an employee works.
A minimal set of relationships can be as follows:
Works_in : employee works in a department
Has_acquired: employee has acquired a skill
Assigned_to : employee assigned to a project
Controlled_by: A project is controlled by a department
Project_manager: Employee is a project_manager of a Project
Solve the Following Queries:

1. List the names of employees in department "... ..... "
2. List the projects along with their properties, controlled by department "....."
3. List the skillset for an employee " "
4. List the names of the projects managed by employee " "
5. List the departments along with the count of employees in it.

Creating Label Employee

```
create(Employee:e1{name:'Sameer',age:28,addrs:'Pune'}) return(Employee)

create(Employee:e2{name:'Seema',age:25,addrs:'Pune'}) return(Employee)

create(Employee:e3{name:'Nisha',age:26,addrs:'Mumbai'}) return(Employee)

create(Employee:e4{name:'Neeta',age:27,addrs:'Nagar'}) return(Employee)
```

Creating Label Department

```
create(Department:d1{id:1,name:'Computer Science',location:'First Floor'}) return Depart
ment

create(Department:d2{id:2,name:'Account',location:'First Floor'}) return Department

create(Department:d3{id:3,name:'Math',location:'M1'}) return Department

create(Department:d4{id:4,name:'Chem',location:'Second Floor'}) return Department
```

Creating Label Skillset

```
create(Skillset:s1{name:'s1',skills:['Communication','Reasoning','Verbal']}) return Skillset

create(Skillset:s2{name:'s2',skills:['Communication','Reasoning']}) return Skillset

create(Skillset:s3{name:'s3',skills:['Communication','Reasoning','Logical']}) return Skillset

create(Skillset:s4{name:'s4',skills:['Communication','Logical']}) return Skillset
```

Creating Label Project

```
create(Project:p1{name:'Robotics',duration:3}) return Project

create(Project:p2{name:'Alexa',duration:4}) return Project

create(Project:p3{name:'BMS',duration:2}) return Project

create(Project:p4{name:'HMS',duration:3}) return Project
```

Creating Relationships:

Works_in : employee works in a department

```
Match(a:e1{name:'Sameer'}),(b:d1{id:1}) create (a)-[r1:Works_in]->(b) return r1

Match(a:e2{name:'Seema'}),(b:d2{id:2}) create (a)-[r2:Works_in]->(b) return r2

Match(a:e3{name:'Nisha'}),(b:d2{id:2}) create (a)-[r3:Works_in]->(b) return r3

Match(a:e3{name:'Nisha'}),(b:d3{id:3}) create (a)-[r4:Works_in]->(b) return r4

Match(a:e4{name:'Neeta'}),(b:d4{id:4}) create (a)-[r5:Works_in]->(b) return r5
```

Has_acquired: employee has acquired a skill

```
Match(a:e4{name:'Neeta'}),(b:s1{name:'s1'}) create (a)-[r1:Has_acquired]->(b) return r1

Match(a:e3{name:'Nisha'}),(b:s3{name:'s3'}) create (a)-[r2:Has_acquired]->(b) return r2

Match(a:e4{name:'Neeta'}),(b:s4{name:'s4'}) create (a)-[r3:Has_acquired]->(b) return r3

Match(a:e2{name:'Seema'}),(b:s2{name:'s2'}) create (a)-[r4:Has_acquired]->(b) return r4
```

Assigned_to : employee assigned to a project

```
Match(a:e1{name:'Sameer'}),(b:p2{name:'Alexa'}) create (a)-[r1:AssignedTo]->(b) return r1

Match(a:e1{name:'Sameer'}),(b:p1{name:'Robotics'}) create (a)-[r2:AssignedTo]->(b) return r2

Match(a:e3{name:'Nisha'}),(b:p3{name:'BMS'}) create (a)-[r3:AssignedTo]->(b) return r3

Match(a:e4{name:'Neeta'}),(b:p4{name:'HMS'}) create (a)-[r4:AssignedTo]->(b) return r4
```

Controlled_by: A project is controlled by a department

```
Match (a:p1{name:'Robotics'}),(b:d1{name:'Computer Science' }) create (a)-[r1:ControlledBy]-
>(b) return r1

Match (a:p2{name:'Alexa'}),(b:d3{name:'Mathematics' }) create (a)-[r2:ControlledBy]-
>(b) return r2

Match (a:p3{name:'BMS'}),(b:d3{name:'Mathematics' }) create (a)-[r3:ControlledBy]-
>(b) return r3

Match (a:p4{name:'HMS'}),(b:d2{name:'Account' }) create (a)-[r4:ControlledBy]->(b) return r4
```

Project_manager: Employee is a project_manager of a Project

```
Match(a:e1{name:'Sameer'}),(b:p2{name:'Alexa'}) create (a)-[r1:Project_Manager_of]-
>(b) return r1

Match(a:e1{name:'Sameer'}),(b:p1{name:'Robotics'}) create (a)-[r2:Project_Manager_of]-
>(b) return r2

Match(a:e3{name:'Nisha'}),(b:p3{name:'BMS'}) create (a)-[r3:Project_Manager_of]-
>(b) return r3

Match(a:e4{name:'Neeta'}),(b:p4{name:'HMS'}) create (a)-[r4:Project_Manager_of]-
>(b) return r4
```

Quries :

1. **List the names of employees in department “Account”**

Query:

```
Match (a)-[:Works_in]->(:d2{name:'Account'}) return a.name
```

Output:

```
employee$ Match (a)-[:Works_in]→(:d2{name:'Account'}) return a.name
```

	a.name
1	"Nisha"
2	"Seema"
3	"Seema"

2. **List the projects along with their properties, controlled by department “Mathematics”**

Query:

```
Match (a)-[:ControlledBy]->(b:d3{name:'Mathematics'}) return a.name
```

Output:

```
employee$ Match (a)-[:ControlledBy]→(b:d3{name:'Mathematics'})
          return a.name
```

	a.name
1	"BMS"
2	"Alexa"
3	"Alexa"

3. **List the skillset for an employee “Nisha”**

Query:

```
Match (a:e3{name:'Nisha'})-[:Has_acquired]->(b) return b.skills
```

Output:

```
employee$ Match (a:e3{name:'Nisha'})-[:Has_acquired]→(b) return
          b.skills
```

	b.skills
1	["Communication", "Reasoning", "Logical"]

4. **List the names of the projects managed by employee “Sameer”**

Query:

```
Match(a:e1{name:'Sameer'})-[:Project_Manager_of]->(b) return b.name
```

Output:

```
employee$ Match(a:e1{name:'Sameer'})-[:Project_Manager_of]→(b)
          return b.name
```

	b.name
1	"Robotics"
2	"Alexa"
3	"Alexa"

5. **List the departments along with the count of employees in it.**

Query:

```
Match (a)-[:Works_in]->(b) return b.name,count(a)
```

Output:

```
employee$ Match (a)-[:Works_in]→(b) return b.name,count(a)
```

Table | Text | Code

	b.name	count(a)
1	"Computer Science"	2
2	"Account"	3
3	"Math"	1
4	"Mathematics"	1
5	"Chem"	1

3. **Solve the following database using Neo4j.**

Create a library database, as given below.
There are individual books, readers, and authors that are
Present in the library data model. A minimal set of labels are
as follows:
Book: This label includes all the books
Person: This label includes authors, translators, reviewers, Readers**,**
Suppliers and so on
Publisher: This label includes the publishers of books in the database

A set of basic relationships are as follows:
Published By:
Issued By:
Returned By:
Solve the Following Queries:

1. List all readers who have issued either book ["Tinker","Tailor","Soldier","spy"] or "One Man in Havana"
2. List the names of voracious readers in our library
3. Add a property “Number of books issued “ for Mr. Joshi and set its value as the count
4. List the readers who haven’t issued any book.
5. List all people, who have issued a book“……”

Library Database

Person Label

```
create(author:Person{name:"John Le Carre",born:19-10-1932}) return author
```

```
create(reader:Person{name:"Ian"}) return reader
```

```
create(reader:Person{name:"Alan"}) return reader
```

```
create(supplier:Person{name:"Rahul"}) return supplier
```

Book Label

```
create(book1:Book{name:"Story Book",title:["Tinker","Tailor","Soldier","spy"],published:1974,city:"Pune"})
```

```
create(book2:Book{name:"Suspens Story Book",title:"One Man in Havana",published:1958,city:'Mumbai'})
```

Publisher Label

```
create(publisher1:Publisher{name:'Mr.Joshi',City:'Pune'})
```

```
create(publisher2:Publisher{name:'Mr.Mohan',City:'Mumbai'})
```

Relationships

```
match(a:Publisher{name:"Mr.Joshi"}),(b:Book{name:"Story Book"}) create(a)-[r1:published_by]->(b)
```

```
match(a:Publisher{name:"Mr.Joshi"}),(b:Book{name:"Suspens Story Book"}) create(a)-[r2:published_by]->(b)
```

```
match(a:Publisher{name:"Mr.Mohan"}),(b:Book{name:"Story Book"}) create(a)-[r3:published_by]->(b)
```

```
match(a:Person{name:"John Le Carre"}),(b:Book{name:"Stor
y Book"}) create(a)-[r:WROTE]->(b)
```

```
match(a:Person{name:"Alan"}),(b:Book{name:"Story Book"}) c
reate(a)-[r4:Issued_by]->(b)
```

```
match(a:Person{name:"lan"}),(b:Book{name:"Suspens Story B
ook"}) create(a)-[r5:Issued_by]->(b)
```

```
match(a:Person),(b:Book) where a.name="Alan" and b.name="
Story Book" create (a)-[r2:RECOMMENDED{date:'05-07-
2011'}]->(b)
```

```
match(a:Person),(b:Book) where a.name="Alan" and b.name="
Story Book" create (a)-[r3:READ{date:'9-9-2011'}]->(b)
```

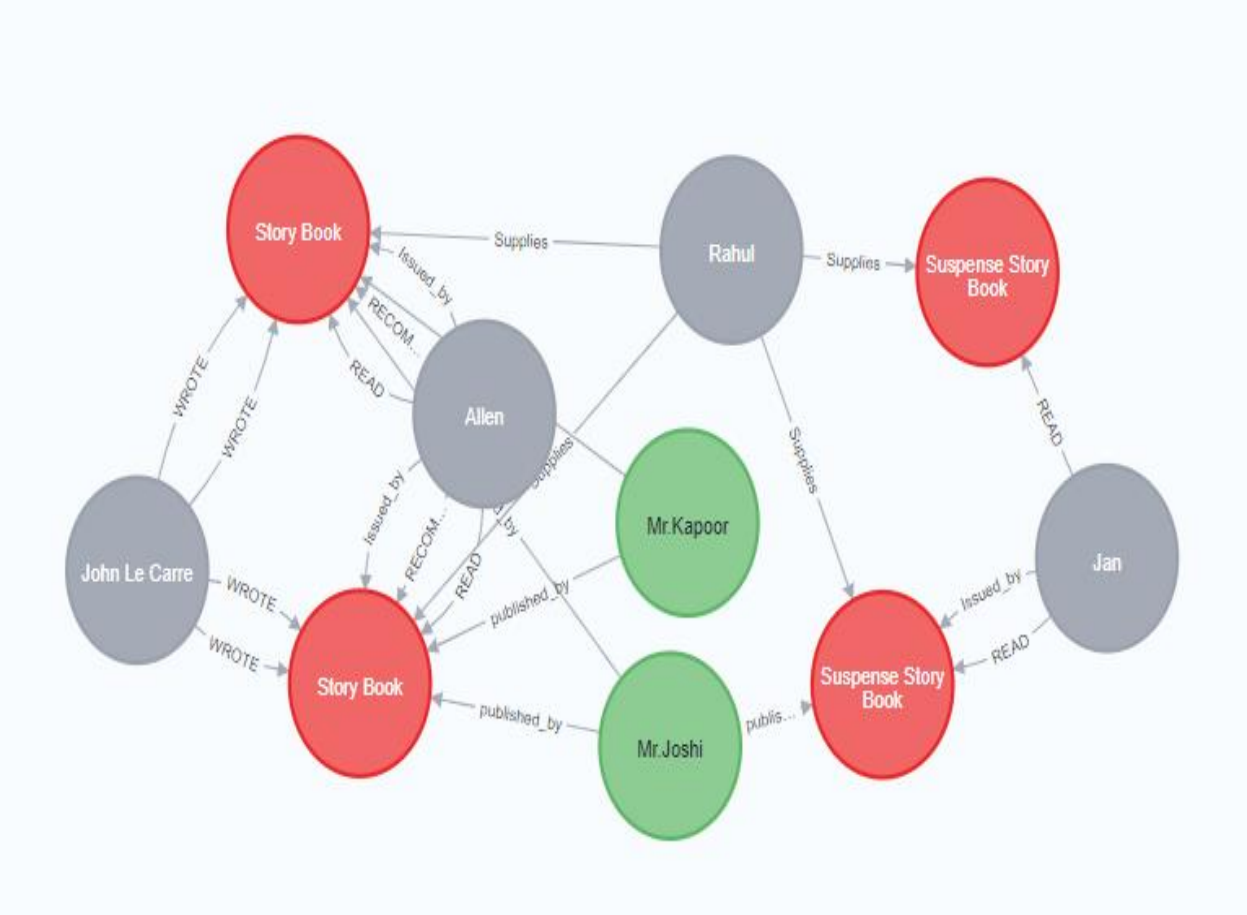
```
match(a:Person),(b:Book) where a.name="lan" and b.name="S
uspens Story Book" create (a)-[r3:READ{date:'9-9-2011'}]-
>(b)
```

```
match(a:Person),(b:Book) where a.name="Rahul" and b.name=
"Story Book" create (a)-[r3:Supplies]->(b)
```

```
match(a:Person),(b:Book) where a.name="Rahul" and b.name=
"Suspens Story Book" create (a)-[r3:Supplies]->(b)
```

```
match (Library) return Library
```

OUTPUT

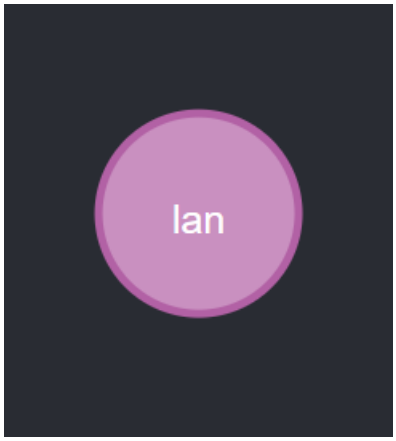


QUERIES

1] List all people, who have issued a book "Our Man in Havana".

```
match(a:Person)-[r1:Issued_by]-
>(b:Book{title:"One Man in Havana"}) return a
```

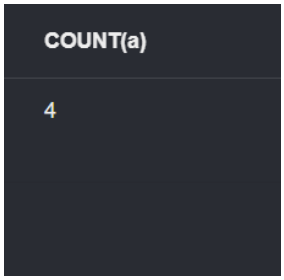
OUTPUT



2] Count the number of people who have read "…."

```
MATCH (a:Person)-[r:READ]-
>(b:Book) WHERE b.title="One Man in Havana" RETURN CO
UNT(a)
```

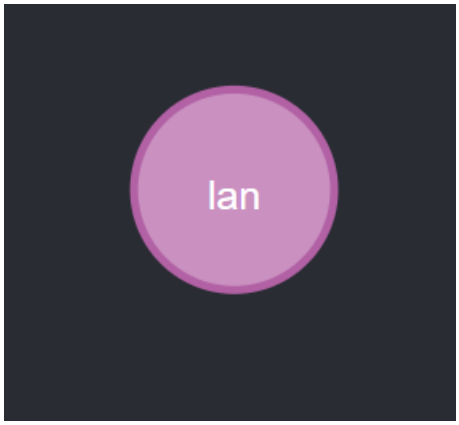
OUTPUT



3] Add a property
"Number of books issued " for Mr. Joshi and set its value as the count

```
MATCH (ir:Person{name:'Ian'}) SET ir.No_of_Issued=1 RET
URN ir
```

OUTPUT



4] List the names of publishers from pune city

```
match(p:Publisher{City:'Pune'}) return p.name
```

p.name
"Mr.Joshi"

5:List all readers who have recommended either book "Tinker","Tailor","Soldier","spy"] or "One Man in Havana"

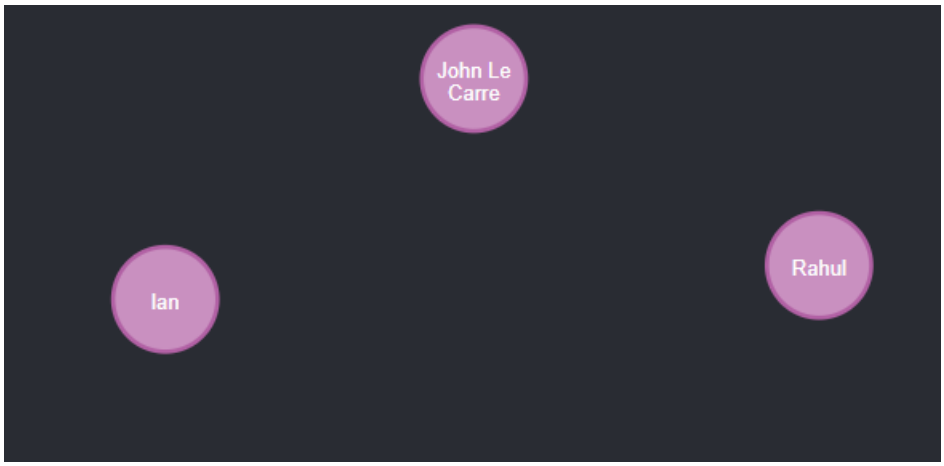
```
MATCH (a:Person)-[r:RECOMMENDED]-
>(b:Book) WHERE b.title=["Tinker","Tailor","Soldier","spy"]
OR b.title="One Man in Havana" RETURN a
```

OUTPUT

6: List the readers who haven't recommended any book

```
MATCH (a:Person) WHERE NOT (a:Person)-
[:RECOMMENDED]->(:Book) return
```

OUTPUT



7: List the reader names who have read/ issued books and display their count.

```
MATCH (a:Person)-[r:Issued_by]-
>(b:Book) RETURN a.name,b.title,COUNT(b)
```

OUTPUT

a.name	b.title
"Ian"	"One Man in Havana"
"Alan"	["Tinker", "Tailor", "Soldier", "spy"]

8: List the names of voracious readers in our library

```
MATCH (a:Person) WHERE NOT(a:Person)-[:Issued_by]->(:Book) RETURN a.name
```

OUTPUT

a.name
"John Le Carre"
"Rahul"