# PPL SLIPS ANSWER

1. Write a program to read an integer from user and convert it to binary and octal using user defined functions.

```
object NumberConversion {

  // Function to convert integer to binary
  def toBinary(n: Int): String = {
  n.toBinaryString
  }

  // Function to convert integer to octal
  def toOctal(n: Int): String = {
  n.toOctalString
  }

  def main(args: Array[String]): Unit = {
    // Reading an integer from the user
    println("Enter an integer: ")
    val number = scala.io.StdIn.readInt()

    // Convert to binary and octal
    val binary = toBinary(number)
    val octal = toOctal(number)

    // Printing the results
    println(s"Binary representation of $number: $binary")
    println(s"Octal representation of $number: $octal")
  }
}
```

1)slip one

```
// Create the graph model
// Create persons CREATE
 (amita:Person {name: 'Amita', age: 28, gender: 'Female', location: 'Pune'}),
```

```
 (rahul:Person {name: 'Rahul', age: 30, gender: 'Male', location: 'Pune'})
, (priya:Person {name: 'Priya', age: 55, gender: 'Female', location: 'Mumbai'}),
 (amit:Person {name: 'Amit', age: 57, gender: 'Male', location: 'Kolhapur'}),
 (neha:Person {name: 'Neha', age: 26, gender: 'Female', location: 'Mumbai'}),
 (ravi:Person {name: 'Ravi', age: 25, gender: 'Male', location: 'Pune'});
 // Create relationships CREATE
 (amita)-[:FRIENDS_WITH]->(neha),
 (amita)-[:HAS_SIBLING]->(ravi),
(priya)-[:PARENT_OF]->(amita),
(amit)-[:PARENT_OF]->(amita),
(rahul)-[:FRIENDS_WITH]->(neha);
 // Query 1: List all mothers
 MATCH  (m:Person)-[:PARENT_OF]->(child:Person)
WHERE m.gender = 'Female' RETURN
m.name AS MotherName;
 // Query 2: List the friends of Amita
 MATCH (amita:Person {name: 'Amita'})-[:FRIENDS_WITH]->(friend:Person)
 RETURN friend.name AS FriendName;
```

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

```
object PerfectNumberCheck {


  // Function to check if a number is perfect
  def isPerfect(n: Int): Boolean = {
    // Finding the sum of divisors
    val sumOfDivisors = (1 until n).filter(n % _ == 0).sum
    sumOfDivisors == n
  }


  def main(args: Array[String]): Unit = {
    // Generating five random numbers
    val randomNumbers = Seq.fill(5)(scala.util.Random.nextInt(500) + 1) // Random numbers between 1 and 500


    // Printing and checking each number
    randomNumbers.foreach { number =>
    println(s"Number: $number")
      if (isPerfect(number)) {
        println(s"$number is a Perfect Number")
      } else {
        println(s"$number is NOT a Perfect Number")
      }
    }
  }
}
```

Slip 2) // Create the graph model

```
// Create persons
CREATE (alice:Person {name: 'Alice', age: 30, location: 'Pune', project: 'Finance'}),
(bob:Person {name: 'Bob', age: 25, location: 'Mumbai', project: 'Sales'}),
(charlie:Person {name: 'Charlie', age: 28, location: 'Kolhapur', project: 'Inventory'}),
(david:Person {name: 'David', age: 35, location: 'Pune', project: 'Finance'}),
(eve:Person {name: 'Eve', age: 22, location: 'Mumbai', project: 'Inventory'}),
(frank:Person {name: 'Frank', age: 27, location: 'Pune', project: 'Sales'});
 // Create friendships
```

```
CREATE  (alice)-[:FRIENDS_WITH]->(david),
 (alice)-[:FRIENDS_WITH]->(frank),
 (bob)-[:FRIENDS_WITH]->(eve),
(david)-[:FRIENDS_WITH]->(frank);
// Query 1: List persons staying in Pune
 MATCH (p:Person {location: 'Pune'})
 RETURN p.name AS PersonName;
 // Query 2: List the persons working on Finance project
 MATCH (p:Person {project: 'Finance'})
RETURN p.name AS PersonName;
```

3. Write a program to check if the matrix is upper triangular or not.

```
object UpperTriangularCheck {

  // Function to check if the matrix is upper triangular
  def isUpperTriangular(matrix: Array[Array[Int]]): Boolean = {
    val rows = matrix.length
    for (i <- 1 until rows) {
      for (j <- 0 until i) {
        if (matrix(i)(j) != 0) {
          return false
        }
      }
    }
    true
  }

  def main(args: Array[String]): Unit = {
    // Example matrix (3x3)
    val matrix = Array(
    Array(1, 2, 3),
      Array(0, 5, 6),
      Array(0, 0, 9)
    )

    // Print the matrix
    println("Matrix:")
    matrix.foreach(row => println(row.mkString(" ")))

    // Check if the matrix is upper triangular
```

```
    if (isUpperTriangular(matrix)) {
      println("The matrix is Upper Triangular")
    } else {
      println("The matrix is NOT Upper Triangular")
    }
  }
}
```

Slip 3) // Create the graph model

```
// Create departments
CREATE (math:Department {name: 'Mathematics', head: 'Dr. Smith'}),
(geo:Department {name: 'Geology', head: 'Dr. Johnson'}),
(chem:Department {name: 'Chemistry', head: 'Dr. Adams'}),
(cs:Department {name: 'Computer Science', head: 'Dr. Lee'});
// Create courses
CREATE (calculus:Course {code: 'MATH101', name: 'Calculus', credits: 4}),
(linearAlgebra:Course {code: 'MATH102', name: 'Linear Algebra', credits: 3}),
(geology101:Course {code: 'GEO101', name: 'Geology 101', credits: 3}),
(organicChemistry:Course {code: 'CHEM201', name: 'Organic Chemistry', credits: 4}),
(dataStructures:Course {code: 'CS101', name: 'Data Structures', credits: 4}),
(algorithms:Course {code: 'CS102', name: 'Algorithms', credits: 4});
// Create relationships
CREATE (math)-[:OFFERS]->(calculus),
(math)-[:OFFERS]->(linearAlgebra),
(geo)-[:OFFERS]->(geology101),
(chem)-[:OFFERS]->(organicChemistry),
(cs)-[:OFFERS]->(dataStructures),
(cs)-[:OFFERS]->(algorithms),
(geo)-[:OFFERS]->(dataStructures); // Cross-department offering
// Query 1: List all the departments of the university
MATCH (d:Department)
RETURN d.name AS DepartmentName;
// Query 2: List the courses provided by the Computer Science department
MATCH (cs:Department {name: 'Computer Science'})-[:OFFERS]->(c:Course)
RETURN c.name AS CourseName, c.code AS CourseCode, c.credits AS Credits;
```

4. Write a program to sort the matrix using insertion sort.

```
object MatrixSort {

 // Function to perform insertion sort on an array
 def insertionSort(arr: Array[Int]): Array[Int] = {
 for (i <- 1 until arr.length) {
    val key = arr(i)
    var j = i - 1

    // Move elements of arr[0..i-1], that are greater than key, to one position ahead
    // of their current position
    while (j >= 0 && arr(j) > key) {
    arr(j + 1) = arr(j)
     j = j - 1
    }
    arr(j + 1) = key
  }
  arr
}

 // Function to flatten a 2D matrix to a 1D array
 def flattenMatrix(matrix: Array[Array[Int]]): Array[Int] = {
   matrix.flatten
 }

 // Function to reshape a 1D array back into a 2D matrix
 def reshapeMatrix(arr: Array[Int], rows: Int, cols: Int): Array[Array[Int]] = {
   arr.grouped(cols).toArray
 }
```

```
  def main(args: Array[String]): Unit = {
    // Example matrix (3x3) val matrix =
    Array( Array(9, 7, 5),
      Array(6, 3, 2),
      Array(8, 1, 4)
    )

    // Print the original matrix println("Original  Matrix:")
    matrix.foreach(row => println(row.mkString(" ")))

    // Flatten the matrix into a 1D array val flatArray =
    flattenMatrix(matrix)

    // Sort the array using insertion sort
    val sortedArray = insertionSort(flatArray)

    // Reshape the sorted array back into a matrix
    val sortedMatrix = reshapeMatrix(sortedArray, matrix.length, matrix(0).length)

    // Print the sorted matrix println("\nSorted Matrix:")
    sortedMatrix.foreach(row => println(row.mkString(" ")))
  }
}
```

Slip 4)
```
// Create departments
CREATE (math:Department {name: 'Mathematics', head: 'Dr. Smith'}),
(chem:Department {name: 'Chemistry', head: 'Dr. Adams'}),
(cs:Department {name: 'Computer Science', head: 'Dr. Lee'}),
(geo:Department {name: 'Geology', head: 'Dr. Johnson'});

 // Create courses
 CREATE (calculus:Course {code: 'MATH101', name: 'Calculus', credits: 4}),
 (organicChemistry:Course {code: 'CHEM201', name: 'Organic Chemistry', credits: 4}), (dataStructures:Course {code:
'CS101', name: 'Data Structures', credits: 4}),
 (algorithms:Course {code: 'CS102', name: 'Algorithms', credits: 4}),
 (geology101:Course {code: 'GEO101', name: 'Geology 101', credits: 3}),
(chemistry101:Course {code: 'CHEM101', name: 'Chemistry 101', credits: 3}), (programmingBasics:Course {code:
'CS103', name: 'Programming Basics', credits: 3}), (chemistryAndEnvironment:Course {code: 'CHEM202', name:
'Chemistry and Environment', credits: 4});

// Create relationships
```

```
 CREATE (chem)-[:OFFERS]->(organicChemistry),
 (chem)-[:OFFERS]->(chemistry101),
 (chem)-[:OFFERS]->(chemistryAndEnvironment),
 (math)-[:OFFERS]->(calculus),
 (cs)-[:OFFERS]->(dataStructures),
 (cs)-[:OFFERS]->(algorithms),
 (cs)-[:OFFERS]->(programmingBasics),
(geo)-[:OFFERS]->(geology101),
(chem)-[:OFFERS]->(organicChemistry),
 (cs)-[:OFFERS]->(chemistryAndEnvironment); // Common course

 // Query 1: List all the courses of the Chemistry department
 MATCH (chem:Department {name: 'Chemistry'})-[:OFFERS]->(course:Course)
 RETURN course.name AS CourseName, course.code AS CourseCode, course.credits AS Credits;

// Query 2: List the common courses across Chemistry and Computer Science departments
 MATCH (chem:Department {name: 'Chemistry'})-[:OFFERS]->(course:Course),
 (cs:Department {name: 'Computer Science'})-[:OFFERS]->(course)
RETURN course.name AS CommonCourseName, course.code AS CourseCode, course.credits AS Credits;
```

5. Write a Scala program to read two strings and perform the following functions on it:

a. Compare using == operator

b. Compare using equals(), compareTo() functions.

c. Find character at position 5.

```
object StringOperations {
  def main(args: Array[String]): Unit = {
    // Reading two strings from the user
    println("Enter the first string: ")
    val string1 = scala.io.StdIn.readLine()

    println("Enter the second string: ")
    val string2 = scala.io.StdIn.readLine()

    // a. Compare using == operator
    println("\nComparing strings using '==' operator:")
    if (string1 == string2) {
      println("Strings are equal")
    } else {
      println("Strings are not equal")
    }

    // b. Compare using equals() and compareTo() methods
    println("\nComparing strings using 'equals()' method:")
    if (string1.equals(string2)) {
      println("Strings are equal")
    } else {
      println("Strings are not equal")
```

```
    }

    println("\nComparing strings using 'compareTo()' method:")
    val comparisonResult = string1.compareTo(string2)
    if (comparisonResult == 0) {
      println("Strings are equal")
    } else if (comparisonResult > 0) {
      println("First string is greater than second string")
    } else {
      println("First string is less than second string")
    }

    // c. Find character at position 5
    println("\nCharacter at position 5 in both strings (if available):")

    if (string1.length >= 5) {
      println(s"Character at position 5 in first string: ${string1.charAt(4)}")
    } else {
      println("First string does not have a character at position 5")
    }

    if (string2.length >= 5) {
      println(s"Character at position 5 in second string: ${string2.charAt(4)}")
    } else {
      println("Second string does not have a character at position 5")
    }
  }
}
```

6. Write a Scala program to read two strings and perform the following functions on it: [10 M]

**a.** Concatenate two strings

**b.** Check if first string ends with “la” (endsWith)

**c.** Find the index of character ‘a’ in second string. (indexOf)

```
object StringFunctions {
  def main(args: Array[String]): Unit = {
    // Reading two strings from the user
    println("Enter the first string: ")
    val string1 = scala.io.StdIn.readLine()

    println("Enter the second string: ")
    val string2 = scala.io.StdIn.readLine()

    // a. Concatenate two strings
    val concatenatedString = string1 + string2
    println(s"\nConcatenated String: $concatenatedString")
    // b. Check if first string ends with "la"
    println("\nChecking if the first string ends with 'la':")
    if (string1.endsWith("la")) {
      println(s"Yes, '$string1' ends with 'la'")
    } else {
      println(s"No, '$string1' does not end with 'la'")
    }
    // c. Find the index of character 'a' in second string
    val indexOfA = string2.indexOf('a')
    println(s"\nIndex of 'a' in the second string: $indexOfA")
  }
}
```

```
[
  { "_id": 1, "name": "Penguin Random House", "address": { "street": "123 Book St", "city": "Mumbai",
```

"state": "Maharashtra", "zip": "400001" }, "contact": { "phone": "022-12345678", "email": "contact@penguinrandomhouse.com" }},

{ "_id": 2, "name": "HarperCollins", "address": { "street": "456 Book Ave", "city": "Delhi", "state": "Delhi", "zip": "110001" }, "contact": { "phone": "011-98765432", "email": "contact@harpercollins.com" }},

{ "_id": 3, "name": "Scholastic", "address": { "street": "789 Paper Rd", "city": "Mumbai", "state": "Maharashtra", "zip": "400002" }, "contact": { "phone": "022-87654321", "email": "contact@scholastic.com" }},

{ "_id": 4, "name": "Macmillan", "address": { "street": "987 Page Blvd", "city": "Bangalore", "state": "Karnataka", "zip": "560001" }, "contact": { "phone": "080-12312312", "email": "contact@macmillan.com" }},

{ "_id": 5, "name": "Oxford University Press", "address": { "street": "321 Leaf Dr", "city": "Mumbai", "state": "Maharashtra", "zip": "400003" }, "contact": { "phone": "022-11223344", "email": "contact@oup.com" }},

{ "_id": 6, "name": "Simon & Schuster", "address": { "street": "654 Pen Ln", "city": "Chennai", "state": "Tamil Nadu", "zip": "600001" }, "contact": { "phone": "044-98765432", "email": "contact@simonandschuster.com" }},

{ "_id": 7, "name": "Bloomsbury", "address": { "street": "789 Ink St", "city": "Kolkata", "state": "West Bengal", "zip": "700001" }, "contact": { "phone": "033-12345678", "email": "contact@bloomsbury.com" }},

{ "_id": 8, "name": "Hachette Livre", "address": { "street": "345 Parchment Ave", "city": "Pune", "state": "Maharashtra", "zip": "411001" }, "contact": { "phone": "020-98765432", "email": "contact@hachette.com" }},

{ "_id": 9, "name": "Wiley", "address": { "street": "654 Scroll Ln", "city": "Hyderabad", "state": "Telangana", "zip": "500001" }, "contact": { "phone": "040-12345678", "email": "contact@wiley.com" }},

{ "_id": 10, "name": "Pearson", "address": { "street": "987 Book Dr", "city": "Ahmedabad", "state": "Gujarat", "zip": "380001" }, "contact": { "phone": "079-98765432", "email": "contact@pearson.com" }}

]

[

{ "_id": 101, "title": "Introduction to MongoDB", "author": "John Doe", "publisher_id": 1, "price": 1200, "ISBN": "978-3-16-148410-0" },

{ "_id": 102, "title": "Advanced Python", "author": "Jane Smith", "publisher_id": 2, "price": 800, "ISBN": "978-0-13-110362-7" },

{ "_id": 103, "title": "Data Science with R", "author": "Alice Johnson", "publisher_id": 3, "price": 1500, "ISBN": "978-1-59327-950-9" },

{ "_id": 104, "title": "AI and Machine Learning", "author": "Bob Brown", "publisher_id": 1, "price": 2500, "ISBN": "978-0-262-03384-8" },

{ "_id": 105, "title": "Cloud Computing Basics", "author": "Carol White", "publisher_id": 4, "price": 600, "ISBN": "978-0-12-802929-9" },

{ "_id": 106, "title": "Full Stack Web Development", "author": "David Green", "publisher_id": 5, "price": 1800, "ISBN": "978-0-321-87814-5" },

{ "_id": 107, "title": "Blockchain Technology", "author": "Eve Black", "publisher_id": 1, "price": 1100, "ISBN": "978-0-387-48935-0" },

{ "_id": 108, "title": "Big Data Analytics", "author": "Frank Blue", "publisher_id": 2, "price": 1300, "ISBN": "978-0-672-33831-6" },

{ "_id": 109, "title": "Quantum Computing Explained", "author": "Grace Red", "publisher_id": 3, "price": 1700, "ISBN": "978-0-19-852925-0" },

{ "_id": 110, "title": "Cybersecurity Essentials", "author": "Harry Grey", "publisher_id": 5, "price": 950, "ISBN": "978-1-891830-75-4" }

]

**Query i) List the publishers in Mumbai**

```
db.publishers.find({ "address.city": "Mumbai" })
```

**Query ii) List the details of books with a cost > 1000**

```
db.books.find({ price: { $gt: 1000 } })
```

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

```
class CurrentAccount(val accNo: Int, val name: String, var balance: Double, val minBalance: Double) {

 // Method to deposit money into the account
 def deposit(amount: Double): Unit = {
  if (amount > 0) {
   balance += amount
   println(s"Amount deposited: $$ $amount")
  } else {
   println("Invalid deposit amount.")
  }
 }

 // Method to withdraw money from the account
 def withdraw(amount: Double): Unit = {
  if (amount > 0 && (balance - amount >= minBalance)) {
   balance -= amount
   println(s"Amount withdrawn: $$ $amount")
  } else if (amount > 0 && (balance - amount < minBalance)) {
   println(s"Insufficient balance! Minimum balance must be maintained: $$ $minBalance")
  } else {
   println("Invalid withdrawal amount.")
  }
 }

 // Method to view the current balance
 def viewBalance(): Unit = {
  println(f"Current Balance: $$ $balance%.2f")
 }
```

```
}

object BankOperations {
  def main(args: Array[String]): Unit = {
    // Creating a CurrentAccount object
    val account = new CurrentAccount(accNo = 123456, name = "John Doe", balance = 5000.0,
minBalance = 1000.0)

    // Perform operations
    println(s"Account No: ${account.accNo}, Name: ${account.name}")
    account.viewBalance()

    account.deposit(2000.0)  // Deposit money
    account.viewBalance()

    account.withdraw(1000.0) // Withdraw money
    account.viewBalance()

    account.withdraw(6000.0)  // Attempt to withdraw more than allowed
    account.viewBalance()
  }
}
```

1)slip one
// Create the graph model // Create persons CREATE (amita:Person {name: 'Amita', age: 28, gender: 'Female', location: 'Pune'}), (rahul:Person {name: 'Rahul', age: 30, gender: 'Male', location: 'Pune'}) , (priya:Person {name: 'Priya', age: 55, gender: 'Female', location: 'Mumbai'}), (amit:Person {name: 'Amit', age: 57, gender: 'Male', location: 'Kolhapur'}), (neha:Person {name: 'Neha', age: 26, gender: 'Female', location: 'Mumbai'}), (ravi:Person {name: 'Ravi', age: 25, gender: 'Male', location: 'Pune'}); // Create relationships CREATE (amita)-[:FRIENDS_WITH]->(neha), (amita)-[:HAS_SIBLING]->(ravi), (priya)-[:PARENT_OF]->(amita), (amit)-[:PARENT_OF]->(amita), (rahul)-[:FRIENDS_WITH]->(neha); // Query 1: List all mothers MATCH (m:Person)-[:PARENT_OF]->(child:Person) WHERE m.gender = 'Female' RETURN m.name AS MotherName; // Query 2: List the friends of Amita MATCH (amita:Person {name: 'Amita'})-[:FRIENDS_WITH]->(friend:Person) RETURN friend.name AS FriendName;

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

```
class Employee(var id: Int, var name: String, var salary: Double) {

  // Method to accept employee details
  def accept(id: Int, name: String, salary: Double): Unit = {
    this.id = id
    this.name = name
    this.salary = salary
  }

  // Method to display employee details
  def display(): Unit = {
    println(s"ID: $id, Name: $name, Salary: $$ $salary")
  }
}

object EmployeeOperations {
  def main(args: Array[String]): Unit = {
    // Create an array of Employee objects
    val employees = Array(
      new Employee(0, "", 0.0),  // Placeholder for employee 1
      new Employee(0, "", 0.0),  // Placeholder for employee 2
      new Employee(0, "", 0.0)   // Placeholder for employee 3
    )

    // Accept employee details
    employees(0).accept(1, "Alice", 50000.0)
    employees(1).accept(2, "Bob", 60000.0)
    employees(2).accept(3, "Charlie", 55000.0)
```

```
    // Display all employees
    println("All Employees:")
    employees.foreach(_.display())

    // Find and display employee with the maximum salary
    val maxSalaryEmployee = employees.maxBy(_.salary)
    println("\nEmployee with the maximum salary:")
    maxSalaryEmployee.display()
  }
}
```

Slip 2)

// Create the graph model // Create persons CREATE (alice:Person {name: 'Alice', age: 30, location: 'Pune', project: 'Finance'}), (bob:Person {name: 'Bob', age: 25, location: 'Mumbai', project: 'Sales'}), (charlie:Person {name: 'Charlie', age: 28, location: 'Kolhapur', project: 'Inventory'}), (david:Person {name: 'David', age: 35, location: 'Pune', project: 'Finance'}), (eve:Person {name: 'Eve', age: 22, location: 'Mumbai', project: 'Inventory'}), (frank:Person {name: 'Frank', age: 27, location: 'Pune', project: 'Sales'}); // Create friendships CREATE (alice)-[:FRIENDS_WITH]->(david), (alice)-[:FRIENDS_WITH]->(frank), (bob)-[:FRIENDS_WITH]->(eve), (david)-[:FRIENDS_WITH]->(frank); // Query 1: List persons staying in Pune MATCH (p:Person {location: 'Pune'}) RETURN p.name AS PersonName; // Query 2: List the persons working on Finance project MATCH (p:Person {project: 'Finance'}) RETURN p.name AS PersonName;

9. Define a class Sports (id, name, description, amount). Derive two classes Indoor and Outdoor. Define appropriate constructors and operations. Create an object and perform operations.

```
// Base class Sports
class Sports(val id: Int, val name: String, val description: String, val amount: Double) {

  // Method to display basic information about the sport
  def displayInfo(): Unit = {
    println(s"Sport ID: $id")
    println(s"Sport Name: $name")
    println(s"Description: $description")
    println(f"Amount: $$ $amount%.2f")
  }
}

// Derived class Indoor
class Indoor(id: Int, name: String, description: String, amount: Double, val location: String)
  extends Sports(id, name, description, amount) {

  // Additional method specific to indoor sports
  def displayIndoorInfo(): Unit = {
    displayInfo()
    println(s"Indoor Sport Location: $location")
  }
}

// Derived class Outdoor
class Outdoor(id: Int, name: String, description: String, amount: Double, val weatherRequirement:
String)
  extends Sports(id, name, description, amount) {

  // Additional method specific to outdoor sports
  def displayOutdoorInfo(): Unit = {
```

```
    displayInfo()
    println(s"Weather Requirement: $weatherRequirement")
  }
}

object SportsOperations {
  def main(args: Array[String]): Unit = {
    // Creating an Indoor sports object
    val tableTennis = new Indoor(1, "Table Tennis", "Indoor game played on a table", 150.0, "Indoor Arena")

    // Creating an Outdoor sports object
    val football = new Outdoor(2, "Football", "Outdoor team sport played on a field", 250.0, "Clear weather")

    // Perform operations and display info
    println("Indoor Sport Details:")
    tableTennis.displayIndoorInfo()

    println("\nOutdoor Sport Details:")
    football.displayOutdoorInfo()
  }
}
```

Slip 3)

// Create the graph model // Create departments CREATE (math:Department {name: 'Mathematics', head: 'Dr. Smith'}), (geo:Department {name: 'Geology', head: 'Dr. Johnson'}), (chem:Department {name: 'Chemistry', head: 'Dr. Adams'}), (cs:Department {name: 'Computer Science', head: 'Dr. Lee'}); // Create courses CREATE (calculus:Course {code: 'MATH101', name: 'Calculus', credits: 4}), (linearAlgebra:Course {code: 'MATH102', name: 'Linear Algebra', credits: 3}), (geology101:Course {code: 'GEO101', name: 'Geology 101', credits: 3}), (organicChemistry:Course {code: 'CHEM201', name: 'Organic Chemistry', credits: 4}), (dataStructures:Course {code: 'CS101', name: 'Data Structures', credits: 4}), (algorithms:Course {code: 'CS102', name: 'Algorithms', credits: 4}); // Create relationships CREATE (math)-[:OFFERS]->(calculus), (math)-[:OFFERS]->(linearAlgebra), (geo)-[:OFFERS]->(geology101), (chem)-[:OFFERS]->(organicChemistry), (cs)-[:OFFERS]->(dataStructures), (cs)-[:OFFERS]->(algorithms), (geo)-[:OFFERS]->(dataStructures); // Cross-department offering // Query 1: List all the departments of the university MATCH (d:Department) RETURN d.name AS DepartmentName; // Query 2: List the courses provided by the Computer Science department MATCH (cs:Department {name: 'Computer Science'})-[:OFFERS]->(c:Course) RETURN c.name AS CourseName, c.code AS CourseCode, c.credits AS Credits;

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

```
object ListGenerator {
  def main(args: Array[String]): Unit = {
    // Function to generate the value for each n using the formula 3n^2 + 4n + 6
    def calculate(n: Int): Int = {
      3 * n * n + 4 * n + 6
    }

    // Create a list by applying the function to values from 1 to 10
    val list = (1 to 10).map(calculate).toList

    // Print the generated list
    println("Generated List: " + list.mkString(", "))
  }
}
```

Slip 4)

// Create departments CREATE (math:Department {name: 'Mathematics', head: 'Dr. Smith'}), (chem:Department {name: 'Chemistry', head: 'Dr. Adams'}), (cs:Department {name: 'Computer Science', head: 'Dr. Lee'}), (geo:Department {name: 'Geology', head: 'Dr. Johnson'}); // Create courses CREATE (calculus:Course {code: 'MATH101', name: 'Calculus', credits: 4}), (organicChemistry:Course {code: 'CHEM201', name: 'Organic Chemistry', credits: 4}), (dataStructures:Course {code: 'CS101', name: 'Data Structures', credits: 4}), (algorithms:Course {code: 'CS102', name: 'Algorithms', credits: 4}), (geology101:Course {code: 'GEO101', name: 'Geology 101', credits: 3}), (chemistry101:Course {code: 'CHEM101', name: 'Chemistry 101', credits: 3}), (programmingBasics:Course {code: 'CS103', name: 'Programming Basics', credits: 3}), (chemistryAndEnvironment:Course {code: 'CHEM202', name: 'Chemistry and Environment', credits: 4}); // Create relationships CREATE (chem)-[:OFFERS]->(organicChemistry), (chem)-[:OFFERS]->(chemistry101), (chem)-[:OFFERS]->(chemistryAndEnvironment), (math)-[:OFFERS]->(calculus), (cs)-[:OFFERS]->(dataStructures), (cs)-[:OFFERS]->(algorithms), (cs)-[:OFFERS]->(programmingBasics), (geo)-[:OFFERS]->(geology101), (chem)-[:OFFERS]->(organicChemistry), (cs)-[:OFFERS]->(chemistryAndEnvironment); // Common course // Query 1: List all the courses of the Chemistry department MATCH (chem:Department {name: 'Chemistry'})-[:OFFERS]->(course:Course) RETURN course.name AS CourseName, course.code AS CourseCode, course.credits AS Credits; // Query 2: List the common courses across Chemistry and Computer Science departments MATCH (chem:Department {name: 'Chemistry'})-[:OFFERS]->(course:Course), (cs:Department {name: 'Computer Science'})-[:OFFERS]->(course) RETURN course.name AS CommonCourseName, course.code AS CourseCode, course.credits AS Credits;

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

```
object StudentMap {
  def main(args: Array[String]): Unit = {
    // Creating a Map with Rollno and FirstName
    val students: Map[Int, String] = Map(
      101 -> "Alice",
      102 -> "Bob",
      103 -> "Alice",
      104 -> "Charlie",
      105 -> "Bob",
      106 -> "David"
    )

    // Grouping students by FirstName
    val groupedStudents: Map[String, List[Int]] = students.toList
      .groupBy { case (rollNo, firstName) => firstName }  // Group by FirstName
      .map { case (firstName, rollNos) => firstName -> rollNos.map(_._1) } // Keep only RollNo

    // Print all student information with the same FirstName
    println("Students with the same FirstName:")
    groupedStudents.foreach { case (firstName, rollNos) =>
    if (rollNos.size > 1) {
        println(s"FirstName: $firstName, Roll Numbers: ${rollNos.mkString(", ")}")
      }
    }
  }
}
```

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

```
object SetOperations {
  def main(args: Array[String]): Unit = {
    // Define two sets of integers
    val set1: Set[Int] = Set(1, 2, 3, 4, 5)
    val set2: Set[Int] = Set(4, 5, 6, 7, 8)

    // Merge the two sets
    val mergedSet: Set[Int] = set1 union set2

    // Print the merged set
    println(s"Merged Set: ${mergedSet.mkString(", ")}")

    // Calculate product of all elements
    val product: Int = mergedSet.product
    println(s"Product of all elements: $product")

    // Calculate average of all elements
    val average: Double = if (mergedSet.nonEmpty) {
      mergedSet.sum.toDouble / mergedSet.size
    } else {
      0.0
    }
    println(f"Average of all elements: $average%.2f")
  }
}
```

```
[
  { "_id": 1, "name": "Penguin Random House", "address": { "street": "123 Book St", "city": "Mumbai",
"state": "Maharashtra", "zip": "400001" }, "contact": { "phone": "022-12345678", "email":
"contact@penguinrandomhouse.com" }},
  { "_id": 2, "name": "HarperCollins", "address": { "street": "456 Book Ave", "city": "Delhi", "state": "Delhi",
"zip": "110001" }, "contact": { "phone": "011-98765432", "email": "contact@harpercollins.com" }},
  { "_id": 3, "name": "Scholastic", "address": { "street": "789 Paper Rd", "city": "Mumbai", "state":
```

"Maharashtra", "zip": "400002" }, "contact": { "phone": "022-87654321", "email": "contact@scholastic.com" }},

{ "_id": 4, "name": "Macmillan", "address": { "street": "987 Page Blvd", "city": "Bangalore", "state": "Karnataka", "zip": "560001" }, "contact": { "phone": "080-12312312", "email": "contact@macmillan.com" }},

{ "_id": 5, "name": "Oxford University Press", "address": { "street": "321 Leaf Dr", "city": "Mumbai", "state": "Maharashtra", "zip": "400003" }, "contact": { "phone": "022-11223344", "email": "contact@oup.com" }},

{ "_id": 6, "name": "Simon & Schuster", "address": { "street": "654 Pen Ln", "city": "Chennai", "state": "Tamil Nadu", "zip": "600001" }, "contact": { "phone": "044-98765432", "email": "contact@simonandschuster.com" }},

{ "_id": 7, "name": "Bloomsbury", "address": { "street": "789 Ink St", "city": "Kolkata", "state": "West Bengal", "zip": "700001" }, "contact": { "phone": "033-12345678", "email": "contact@bloomsbury.com" }},

{ "_id": 8, "name": "Hachette Livre", "address": { "street": "345 Parchment Ave", "city": "Pune", "state": "Maharashtra", "zip": "411001" }, "contact": { "phone": "020-98765432", "email": "contact@hachette.com" }},

{ "_id": 9, "name": "Wiley", "address": { "street": "654 Scroll Ln", "city": "Hyderabad", "state": "Telangana", "zip": "500001" }, "contact": { "phone": "040-12345678", "email": "contact@wiley.com" }},

{ "_id": 10, "name": "Pearson", "address": { "street": "987 Book Dr", "city": "Ahmedabad", "state": "Gujarat", "zip": "380001" }, "contact": { "phone": "079-98765432", "email": "contact@pearson.com" }}

]

[

{ "_id": 101, "title": "Introduction to MongoDB", "author": "John Doe", "publisher_id": 1, "price": 1200, "ISBN": "978-3-16-148410-0" },

{ "_id": 102, "title": "Advanced Python", "author": "Jane Smith", "publisher_id": 2, "price": 800, "ISBN": "978-0-13-110362-7" },

{ "_id": 103, "title": "Data Science with R", "author": "Alice Johnson", "publisher_id": 3, "price": 1500, "ISBN": "978-1-59327-950-9" },

{ "_id": 104, "title": "AI and Machine Learning", "author": "Bob Brown", "publisher_id": 1, "price": 2500, "ISBN": "978-0-262-03384-8" },

{ "_id": 105, "title": "Cloud Computing Basics", "author": "Carol White", "publisher_id": 4, "price": 600, "ISBN": "978-0-12-802929-9" },

{ "_id": 106, "title": "Full Stack Web Development", "author": "David Green", "publisher_id": 5, "price": 1800, "ISBN": "978-0-321-87814-5" },

{ "_id": 107, "title": "Blockchain Technology", "author": "Eve Black", "publisher_id": 1, "price": 1100, "ISBN": "978-0-387-48935-0" },

{ "_id": 108, "title": "Big Data Analytics", "author": "Frank Blue", "publisher_id": 2, "price": 1300, "ISBN": "978-0-672-33831-6" },

{ "_id": 109, "title": "Quantum Computing Explained", "author": "Grace Red", "publisher_id": 3, "price": 1700, "ISBN": "978-0-19-852925-0" },

{ "_id": 110, "title": "Cybersecurity Essentials", "author": "Harry Grey", "publisher_id": 5, "price": 950, "ISBN": "978-1-891830-75-4" }

]

**Query i) List the publishers in Mumbai**

```
db.publishers.find({ "address.city": "Mumbai" })
```

**Query ii) List the details of books with a cost > 1000**

```
db.books.find({ price: { $gt: 1000 } })
```