**Candidate Name: Date:**

**Programming Questions:**

1. Write a program which will find numbers which are divisible by 7 but not a multiple of 5 between 2000 and 3200 (both included). The numbers obtained should be printed in a comma-separated sequence on a single line.
2. Write a program to find largest sequence in a given list of numbers

INPUT = [1,2,3,2,4,5,6,7,8,1,0,4,5,6]

Expected Result: [4, 5, 6, 7, 8]

1. Write a program to print the possible combinations of any given word

INPUT = "CAT"

Expected Result: "CAT","CTA","ACT","ATC","TCA","TAC"

**SQL Questions:**

**Question 1:**

**Table Name: Transactions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Account\_number** | **transaction\_time** | **Transaction\_id** | **balance** |
| 123 | 1/1/2019 8:00 | 101 | 1000 |
| 123 | 1/2/2019 8:00 | 102 | 2000 |
| 123 | 1/3/2019 8:00 | 103 | 3000 |
| 789 | 1/4/2019 8:00 | 104 | 1000 |
| 789 | 1/5/2019 8:00 | 105 | 500 |
| 123 | 1/6/2019 8:00 | 106 | 4000 |

Using the above (Transactions) dataset, come up with a SQL query to get the most recent/latest balance, transaction for each account number.

|  |  |  |
| --- | --- | --- |
| **Account\_number** | **Transaction\_id** | **balance** |
| 123 | 106 | 4000 |
| 789 | 105 | 500 |

**Expected Output**

**Question 2:**

**Table Name : Student Table Name : Student\_info**

|  |  |  |
| --- | --- | --- |
| **Studentid** | **Subject** | **Marks** |
| 1 | English | 80 |
| 1 | Maths | 70 |
| 1 | Science | 75 |
| 1 | English | 85 |
| 2 | English | 35 |
| 3 | English | 100 |
| 3 | Maths | 39 |

|  |  |  |
| --- | --- | --- |
| **Studentid** | **DOB** | **Location** |
| 1 | 1/1/1993 | Chennai |
| 3 | 1/1/1993 | Bangalore |

**Expected output**

|  |
| --- |
| **Studentid** |
| 2 |

Using the above 2 tables, Write a SQL to get the list of students, for whom student\_info table doesn’t have data?

**Note**: without using “not in” Clause

**Question 3**

|  |  |  |  |
| --- | --- | --- | --- |
| **Product Code** | **Quantity** | **Inventory Date** | **Cumulative Sum (output)** |
| P1 | 10 | 1 Aug 2019 | 10 |
| P1 | 50 | 2 Aug 2019 | 60 |
| P1 | 20 | 3 Aug 2019 | 80 |
| P1 | 10 | 4 Aug 2019 | 90 |
| P2 | 200 | 1 Aug 2019 | 200 |
| P2 | 350 | 2 Aug 2019 | 550 |
| P2 | 400 | 3 Aug 2019 | 950 |
| P3 | 600 | 1 Aug 2019 | 600 |
| P3 | 250 | 2 Aug 2019 | 850 |

**Product Inventory Table**

Using the above inventory dataset, Write a SQL query to get the cumulative sum (mentioned above) ordered by inventory date.

**Answers:**

**1) Scala**

**=========**

**import scala.collection.mutable.ArrayBuffer**

**val listOfResult=ArrayBuffer[Int]()**

**for(i <- 2300 to 3200)**

**{**

**if((i%7==0) && (i%5!=0))**

**listOfResult+=i**

**}**

**println(listOfResult.toString.substring(listOfResult.toString.indexOf("(")+1,listOfResult.toString.indexOf(")")))**

**2) Spark**

**=========**

**import org.apache.spark.sql.functions.\_**

**import org.apache.spark.sql.expressions.\_**

**import scala.util.control.Breaks.\_**

**import scala.collection.mutable.ArrayBuffer**

**import scala.collection.mutable.ListBuffer**

**case class resultArrayAndCount(resultArray:List[Int],resultArraySize:Int)**

**val resultListBuffer=new ListBuffer[resultArrayAndCount]()**

**val inputList=List(1,2,3,2,4,5,6,7,8,1,0,4,5,6)**

**val numIterations=inputList.size**

**for(iteration <- 1 to numIterations-1)**

**{**

**var indexNoGlobal=0**

**//println("iteration number="+iteration)**

**breakable**

**{**

**val listOfResult=ArrayBuffer[Int]()**

**for( indexNo <- iteration-1 to numIterations-1) // array index is one less**

**{**

**indexNoGlobal=indexNo**

**//println("indexNo number="+indexNo)**

**if(indexNo == numIterations-1)**

**{**

**//println("inputList(indexNo)-1 = "+(inputList(indexNo)-1))**

**//println("inputList(indexNo-1) = "+(inputList(indexNo-1)))**

**if(inputList(indexNo)-1 == inputList(indexNo-1))**

**listOfResult+=inputList(indexNo)**

**else**

**{**

**//println(listOfResult)**

**resultListBuffer+=resultArrayAndCount(inputList.slice(iteration-1,indexNo),listOfResult.size)**

**break**

**}**

**}**

**if(indexNo < numIterations-1)**

**{**

**//println("inputList(indexNo)+1 = "+(inputList(indexNo)+1))**

**//println("inputList(indexNo+1) = "+(inputList(indexNo+1)))**

**if(inputList(indexNo)+1 == inputList(indexNo+1))**

**listOfResult+=inputList(indexNo)**

**else**

**{**

**println(listOfResult)**

**println(inputList.slice(iteration-1,listOfResult.size))**

**resultListBuffer+=resultArrayAndCount(inputList.slice(iteration-1,indexNo),listOfResult.size)**

**break**

**}**

**}**

**}**

**println(listOfResult)**

**resultListBuffer+=resultArrayAndCount(inputList.slice(iteration-1,indexNoGlobal),listOfResult.size)**

**}**

**}**

**val resultTemp=resultListBuffer.toSeq.toDF.orderBy($"resultArraySize".desc).withColumn("row\_num",monotonically\_increasing\_id).filter("row\_num =0").select("resultArray").collect**

**val result=resultTemp(0).toString.substring(resultTemp(0).toString.indexOf("(")+1,resultTemp(0).toString.indexOf(")")).split(",").map(\_.trim).map(\_.toInt)**

**3)spark**

**=========**

**def factTR(number:Int, temp:Int):Int=**

**{**

**if(number ==1)**

**return temp**

**return factTR(number - 1, number \* temp)**

**}**

**def fact(n:Int ) :Int=**

**{**

**return factTR(n, 1)**

**}**

**import java.util.concurrent.ThreadLocalRandom**

**import scala.collection.mutable.ListBuffer**

**//val inputString="CAT"**

**val inputString="dumpty".toUpperCase**

**val inputStringList=inputString.toList**

**val numberOfCombinations=fact(inputStringList.size)**

**val randomListBuffer=ListBuffer[String]()**

**var iterationNumber=1**

**while( iterationNumber<=numberOfCombinations)**

**{**

**var generatedStringSize=1**

**var generationString=""**

**while (generatedStringSize<=inputStringList.size)**

**{**

**val indexOfList=ThreadLocalRandom.current().nextInt(0,inputStringList.size)**

**println("Iteration ="+iterationNumber)**

**println("indexOfList ="+indexOfList)**

**if (!(generationString.contains(inputStringList(indexOfList)))) // omitting if the charecter already exists**

**{**

**generationString=generationString+inputStringList(indexOfList)**

**generatedStringSize=generatedStringSize+1**

**}**

**}**

**if(!(randomListBuffer.contains(generationString))) // omitting if the string combination already exists**

**{**

**randomListBuffer+=generationString**

**iterationNumber=iterationNumber+1**

**}**

**}**

**//result**

**randomListBuffer.toSeq.toDF.show(numberOfCombinations,false)**

**randomListBuffer.toSeq.map(println)**

**4**

**====**

**select Account\_number,Transaction\_id,balance from (select Account\_number,Transaction\_id,balance ,transaction\_time, rank() over( partition by Account\_number order by transaction\_time desc) as rowNum from transaction ) a where rowNum=1**

**spark.read.option("header","true").option("delimiter","|").option("inferSchema","true").csv("file:///home/raptor/IdeaProjects/SparkLearning/Input/temp/rank\_input\_1.txt").createOrReplaceTempView("transaction")**

**spark.sql("select Account\_number,Transaction\_id,balance from (select Account\_number,Transaction\_id,balance ,transaction\_time, rank() over( partition by Account\_number order by transaction\_time desc) as rowNum from transaction ) a where rowNum=1").show**

**5**

**====**

**select distinct a.Studentid from student a left join Student\_info b on a.Studentid=b.Studentid where b.Studentid is null**

**spark.read.option("header","true").option("delimiter","|").option("inferSchema","true").csv("file:///home/raptor/IdeaProjects/SparkLearning/Input/temp/sudent\_test\_1.txt").createOrReplaceTempView("student")**

**spark.read.option("header","true").option("delimiter","|").option("inferSchema","true").csv("file:///home/raptor/IdeaProjects/SparkLearning/Input/temp/student\_info\_1.txt").createOrReplaceTempView("Student\_info")**

**spark.sql("select distinct a.Studentid from student a left join Student\_info b on a.Studentid=b.Studentid where b.Studentid is null").show**

**6**

**=======**

**select Product\_Code,**

**Quantity, Inventory\_Date,**

**sum(Quantity)**

**over ( partition by Product\_Code order by Quantity rows between unbounded preceding and current row ) cum\_sum\_of\_Quantity**

**from Inventory**

**order by Product\_Code,Quantity**

**spark.read.option("header","true").option("delimiter","|").option("inferSchema","true").csv("file:///home/raptor/IdeaProjects/SparkLearning/Input/temp/cumulative\_sum\_eg\_input.txt").createOrReplaceTempView("Inventory")**

**spark.sql("""select Product\_Code,**

**Quantity, Inventory\_Date,**

**sum(Quantity)**

**over ( partition by Product\_Code order by Quantity rows between unbounded preceding and current row ) cum\_sum\_of\_Quantity**

**from Inventory**

**order by Product\_Code,Quantity""").show**

**//val df=spark.read.option("header","true").option("delimiter","|").option("inferSchema","true").csv("file:///home/raptor/IdeaProjects/SparkLearning/Input/temp/cumulative\_sum\_eg\_input.txt")**

**df.withColumn("cumSum",sum("Quantity").over(Window.partitionBy($"Product\_Code").orderBy($"Inventory\_Date"))).orderBy("Product\_Code","Inventory\_Date").show**