Assignment 2 Report

CZ4031: Database Design Principles

Academic Year 2017/2018

Semester 1

GROUP ID: 18

Student Names:	Matric Number:	Tasks Completed:	Individual Contribution:
Yong Guo Jun	U1440217C	Analyzing Query, Translation of Query Plans to Text Descriptions, Readme	35%
Huang Jian Wei	U1521567A	Algorithm, Analyzing Query	30%
See Xin Yee	U1520918B	Implemented Text-to-Speech API and integrate with main Query Plan Converter	20%
Shannon Neo Si Lin	U1521821L	Application Testing	15%

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Table of Contents

1 Introduction

1.1 Background

For this assignment, we are required to program a Java application that can output a set of Query plan with a Query as an input. In addition, the Query plan must be translated into natural language and then be vocalized for the end user. PostgreSQL has been chosen as our DBMS and Eclipse JEE IDE was used to develop our application. On top of that, we made sure our algorithm works in synchronization with DBMS.

1.2 Approach

Firstly, we use Eclipse JEE IDE to code our application. In order to connect to PostgreSQL on our desktop, we have to download PostgreSQL JDBC driver and add it in our Eclipse library to be used. Once we are able to connect to PostgreSQL through our Java application, we will allow user to input queries of their choice and output the query plan into an .xml file format. This is because XML can accurately store the tree structure of the original query plan. The system will then try to read the xml file and extract important statements to form concise statement to be translated to natural language and be vocalized.

2 Algorithm

2.1 Description of Algorithm

To ensure the translated natural language statements are as concise as possible, we have to identify the important keyword and phrases to be kept. Some of the more important operations that we kept are as follows:

Operation	Attributes Kept	Attributes Discarded
Aggregate Operation	Strategy **Example**: The execution of this Query begins by running an Aggregate Operation using **Hashed Strategy** on the result sets of its sub branches.	Partial Mode, Parent Relationship, Parallel Aware, Start-up Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Actual-Total-Time, Actual-Rows, Actual-Loops, Group-Key, Item

Hash Join Operation	Hash-Cond **Example**: In the Sub Branches, **Hash Join Operations** are performed between **authored and authors, authored and books**,**articlesauthored and authors, authored and publications.**	Parent-Relationship, Parallel Aware, Join-Type, Start-up Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Actual-Total-Time, Actual-Rows, Actual-Loops
Seq Scan/Sequential Scan	Relationship-Name, Filter **Example**: **Sequential Scan Operations** are also **performed on authored** tables,**books** tables**,authors** tables.	Parent-Relationship, Parallel Aware, Alias, Start-up Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Actual-Total-Time, Actual-Loops
Index Scan	Index-Name, Relationship Name, Index-Cond, Filter **Example**: Index Scan(s) are used on **authors** table using **authors_pkey key**.	Parent-Relationship, Parallel Aware, Scan-Direction, Alias, Start-up Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Actual-Total-Time, Actual-Loops, Rows-Removed-By-Index-Recheck, Rows-Removed-By-Filter
Sort	Sort-Method **Example**: The method that was used for the **Sort Operation** is **external merge**.	Parent-Relationship, Parallel Aware, Start-up Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Actual-Total-Time, Sort-Space-Used, Sort-Space-Type, Item, Sort-Key, Actual Loops, Actual Rows

Bitmap Heap Scan	Relationship-Name **Example**: The table that **Bitmap Heap Scan** Operation was performed on is **publications**.	Parent-Relationship, Parallel Aware, Alias, Start-up Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Exact-Heap-Blocks, Lossy-Heap-Blocks, Rows-Removed-By-Index-Recheck, Recheck-Cond, Actual-Total-Time, Actual-Rows, Actual-Loops
Bitmap Index Scan	Index-Name, Index-Cond **Example**: There are 1 instance of **Bitmap Index Scan** performed during this Query using condition(s) **month = 'July'**.	Parent-Relationship, Parallel Aware, Start-up Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Actual-Total-Time, Actual-Rows, Actual-Loops
Merge Join	Merge Cond Example**: Merge Join(s)** was used with conditons **proceedings."proceedingsID" = publications."pubID".**	Parent-Relationship, Parallel Aware, Join-Type, Start-Up-Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Actual-Total-Time, Actual-Rows, Actual-Loops
Index Only Scan	Relation-Name,Index-Name,Index-Cond **Example:** Index Scan(s) are used on **proceedings table** using **proceedings_pkey key**, **publications** using **publications_pkey** key with **month = 'July'** filter**.**	Parent-Relationship, Parallel Aware, Scan-Direction, Alias, Startup-Cost, Total-Cost, Plan-Rows, Plan-Width, Actual-Startup-Time, Actual-Total-Time, Actual-Rows, Actual-Loops, Heap-Fetches, Rows-Removed-By-Index-Recheck

2.2 Implementation of Algorithm (Source Code)

*Queryplanmain.java TextToSpeechConvertor.java

```java
package qpc;

import java.io.BufferedReader;...

public class Queryplanmain {

    public static void main(String[] args) {
        System.out
                .println("-------- PostgreSQL JDBC Connection Testing ------------");
        // Pathname of XML file to be written to.
        // NOTE To-Change based on different computers
        String FILENAME = "D:\\QueryPlanConverter\\QueryPlan\\QueryPlan.xml";

        // Test if Postgres Drivers are installed correctly
        try {
            Class.forName("org.postgresql.Driver");

        } catch (ClassNotFoundException e) {
            System.out.println("Where is your PostgreSQL JDBC Driver? "
                    + "Include in your library path!");
            e.printStackTrace();
            return;
        }
        System.out.println("PostgreSQL JDBC Driver Registered!");

        // ------------------------------------------------------------ End of
        // Driver Test

        //For SQL queries
        Connection connection = null;
        Statement stmt = null;

        ////////////////
        //Write to XML document
        Document mapDoc = null;
        //Text to speech object using FreeTTS API
        TextToSpeechConvertor ttsc = new TextToSpeechConvertor();
        try {
            DocumentBuilderFactory.newInstance();

        } catch (Exception e) {
            System.out.println("Problem creating document: " + e.getMessage());
        }

        //Try connection to PostgreSQL database, parameter #1= username, parameter#2 = database password
        try {

            connection = DriverManager.getConnection(
                    "jdbc:postgresql://127.0.0.1:5432/CZ4031", "postgres",
                    "123456");

        } catch (SQLException e) {
            System.out.println("Connection Failed! Check output console");
            e.printStackTrace();
            return;

        }
        if (connection != null) {
            System.out.println("Connection Success!");
        } else {
            System.out.println("Failed to make connection!");
        }

```

```java

        while (true) {

            //Enable write to file
            FileWriter fw;
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                    System.in));
            ArrayList resultSet = new ArrayList();
            int counter = 0;
            try {
                // Get query from user
                System.out.println("Enter Query:");
                String query = reader.readLine();
                fw = new FileWriter(FILENAME);
                BufferedWriter bw = new BufferedWriter(fw);
                stmt = connection.createStatement();
                //Execute Query with EXPLAIN (analyze, FORMAT XML) to
                //output in XML format
                ResultSet rs = stmt
                        .executeQuery("EXPLAIN (analyze, FORMAT XML) " + query);
                System.out.println();
                bw.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
                while (rs.next()) {
                    resultSet.add(rs.getSQLXML(1).getString());
                    bw.write(resultSet.get(counter).toString());
                    counter++;
                }
                bw.close();


                File fXmlFile = new File(
                        "D:\\QueryPlanConverter\\QueryPlan\\QueryPlan.xml");
                DocumentBuilderFactory dbf = DocumentBuilderFactory
                        .newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
                Document doc = db.parse(fXmlFile);
                doc.getDocumentElement().normalize();

                String output = "The execution of this Query begins by running ";
                String hashJoin = "";
                String seqScan = "";
                String indexScan = "";
                String bitmapHeapScan = "";
                String bitmapIndexScan = "";
                String bitmapTables = "";
                String mergeJoins = "";
                String indexOnlyScan = "";
                int noOfIndexOnlyScan = 0;
                int noOfMergeJoins = 0;
                int noOfIndexScan = 0;
                int noOfBitmapHeapScanTables = 0;
                int noOfBitmapHeapScan = 0;
                int noOfBitmapIndexScan = 0;
                int root = 1;
                int subTree = 0;
                int seqsScan = 0;
                int hashsJoin = 0;
                String planningTime = "";
                int noOfSort = 0;
                String sortMethod = "";
```

*Queryplanmain.java | TextToSpeechConvertor.java

```
156                 String exeTime = "";
157                 //Store in List to use to check for repeated tables
158                 ArrayList<String> list = new ArrayList<String>();
159                 ArrayList<String> sortlist = new ArrayList<String>();
160                 ArrayList<String> mergelist = new ArrayList<String>();
161
162                 NodeList nodeList = doc.getElementsByTagName("*");
163                 for (int i = 0; i < nodeList.getLength(); i++) {
164                     // Get element
165                     Element element = (Element) nodeList.item(i);
166
167                     if (!element.getNodeName().contains("Plan")
168                             && !element.getNodeName().contains("Triggers")) {
169
170                         if (root == 1) {
171                             if ((element.getNodeName().equals("Node-Type"))) {
172
173                                 if (element.getFirstChild().getNodeValue()
174                                         .equals("Aggregate")) {
175                                     Element temp = (Element) nodeList
176                                             .item(i + 1);
177                                     output += "an Aggregate Operation using "
178                                             + temp.getFirstChild()
179                                                     .getNodeValue()
180                                             + " Strategy on the result sets of its sub branches.";
181                                     root++;
182                                 } else if (element.getFirstChild()
183                                         .getNodeValue().equals("Hash Join")) {
184                                     output += "a Hash Join Operation.";
185                                     root++;
186                                 }
189                                 else if (element.getFirstChild().getNodeValue()
190                                         .equals("Append")) {
191                                     output += "an Append Operation.";
192                                     root++;
193                                 }
194
195                                 else if (element.getFirstChild().getNodeValue()
196                                         .equals("Seq Scan")) {
197                                     output += "a Sequential Scan Operation.";
198                                     root++;
199                                 } else if (element.getFirstChild()
200                                         .getNodeValue().equals("Hash")) {
201                                     output += "a Hash Operation.";
202                                     root++;
203                                 }
204
205                                 else if (element.getFirstChild().getNodeValue()
206                                         .equals("Nested Loop")) {
207
208                                     output += "a Nested Loop Operation.";
209                                     root++;
210                                 } else if (element.getFirstChild()
211                                         .getNodeValue().equals("Index Scan")) {
212                                     output += "an Index Scan Operation.";
213                                     root++;
214                                 } else if (element.getFirstChild()
215                                         .getNodeValue().equals("Sort")) {
216                                     output += "a Sort Operation.";
217                                     root++;
218                                 } else if (element.getFirstChild()
219                                         .getNodeValue()
220                                         .equals("Bitmap Heap Scan")) {
```

```
                                    output += "a Bitmap Heap Scan Operation.";
                                    root++;
                                } else if (element.getFirstChild()
                                        .getNodeValue()
                                        .equals("Bitmap Index Scan")) {
                                    output += "a Bitmap Index Scan Operation.";
                                    root++;
                                } else if (element.getFirstChild()
                                        .getNodeValue().equals("Merge Join")) {
                                    output += "a Merge Join Operation.";
                                    root++;
                                } else if (element.getFirstChild()
                                        .getNodeValue()
                                        .equals("Index Only Scan")) {
                                    output += "an Index Only Scan Operation.";
                                    root++;
                                } else if (element.getFirstChild()
                                        .getNodeValue().equals("Materialize")) {
                                    output += "a Materialize Operation.";
                                    root++;
                                } else if (element.getFirstChild()
                                        .getNodeValue().equals("Unique")) {
                                    output += "an Unique Operation.";
                                    root++;
                                } else if (element.getFirstChild()
                                        .getNodeValue().equals("Limit")) {
                                    output += "a Limit Operation.";
                                    root++;
                                } else if (element.getFirstChild()
                                        .getNodeValue().equals("Subquery Scan")) {
                                    output += "a Subquery Scan Operation.";

                                    root++;
                                } else if (element.getFirstChild()
                                        .getNodeValue().equals("Group")) {
                                    output += "a Group Operation.";
                                    root++;
                                }
                            }
                        }

                        if (root != 1) {// sub branch
                            if (element.getNodeName().equals("Node-Type")) {
                                if (element.getFirstChild().getNodeValue()
                                        .equals("Hash Join")) {
                                    Element temp = (Element) nodeList
                                            .item(i + 12);
                                    String relations = temp.getFirstChild()
                                            .getNodeValue();
                                    relations = relations.replace("\"", "");
                                    relations = relations.replace("(", "");
                                    relations = relations.replace(")", "");
                                    relations = relations.replace(" ", "");
                                    relations = relations.replace("_", "");
                                    relations = relations.replace("1", "");
                                    relations = relations.replace("2", "");
                                    relations = relations.replace("3", "");
                                    relations = relations.replace("4", "");
                                    relations = relations.replace("5", "");
                                    relations = relations.replace("6", "");
                                    relations = relations.replace("7", "");
                                    relations = relations.replace("8", "");
                                    String[] relation = relations.split("=");
                                    String relation1 = relation[0];
```

```
                    String relation2 = relation[1];
                    String[] r1 = relation1.split("\\.");
                    String[] r2 = relation2.split("\\.");

                    if (hashsJoin == 0) {
                        hashJoin += "In the Sub Branches, Hash Join Operations are performed between "
                                + r1[0] + " and " + r2[0];
                        hashsJoin++;
                    } else {
                        hashJoin += "," + r1[0] + " and "
                                + r2[0];
                    }

                }
            }

        }

        if (root != 1) {// sub branch
            boolean skip = false;
            if (element.getNodeName().equals("Node-Type")) {
                if (element.getFirstChild().getNodeValue()
                        .equals("Sort")) {

                    noOfSort++;
                    Element temp = (Element) nodeList
                            .item(i + 14);
                    if (temp.getNodeName().equals(
                            "Sort-Space-Used")) {
                        temp = (Element) nodeList.item(i + 13);
                    }
                    String col = temp.getFirstChild()
                            .getNodeValue();
                    for (int t = 0; t < sortlist.size(); t++) {
                        if ((col.equals(sortlist.get(t)))) {
                            skip = true;
                        }
                    }
                    if (skip != true) {
                        if (noOfSort == 1) {
                            sortMethod += temp.getFirstChild()
                                    .getNodeValue();

                        } else {
                            sortMethod += ","
                                    + temp.getFirstChild()
                                            .getNodeValue();
                        }
                    }
                    sortlist.add(col);
                }
            }
        }
```

```java
                    // Handle Scanning Operations
                    if (root != 1) {// sub branch
                        boolean skip = false;

                        if (element.getNodeName().equals("Node-Type")) {
                            if (element.getFirstChild().getNodeValue()
                                    .equals("Seq Scan")) {
                                Element temp = (Element) nodeList
                                        .item(i + 3);
                                Element filters = (Element) nodeList
                                        .item(i + 13);
                                String filter = filters.getNodeName();

                                String column = temp.getFirstChild()
                                        .getNodeValue();

                                for (int j = 0; j < list.size(); j++) {
                                    if ((column.equals(list.get(j)))) {
                                        skip = true;
                                    }
                                }

                                if (skip != true) {
                                    if (seqsScan == 0 && hashsJoin == 0) {
                                        seqScan += "In the Sub Branches, Sequential Scan Operations are performed on "
                                                + column + " tables";
                                        if (filter.equals("Filter")) {
                                            String filterContent = filters
                                                    .getFirstChild()
                                                    .getNodeValue();
                                            filterContent = filterContent
                                                    .replace("::text", "");
                                            filterContent = filterContent
                                                    .replace("(", "");
                                            filterContent = filterContent
                                                    .replace(")", "");
                                            filterContent = filterContent
                                                    .replace("&gt;=", "");
                                            filterContent = filterContent
                                                    .replace("&gt;=", "");
                                            seqScan += " using filter ("
                                                    + filterContent + ")";
                                        }

                                        list.add(column);
                                        seqsScan++;
                                    } else if (seqsScan == 0
                                            && hashsJoin != 0) {
                                        seqScan += "Sequential Scan Operations are also performed on "
                                                + column + " tables";
                                        if (filter.equals("Filter")) {
                                            String filterContent = filters
                                                    .getFirstChild()
                                                    .getNodeValue();
                                            filterContent = filterContent
                                                    .replace("::text", "");
                                            filterContent = filterContent
                                                    .replace("(", "");
                                            filterContent = filterContent
                                                    .replace(")", "");
                                            filterContent = filterContent
```

```
                                        filterContent = filterContent
                                                .replace("&gt;=", "");
                                        filterContent = filterContent
                                                .replace("&gt;=", "");
                                        seqScan += " using filter ("
                                                + filterContent + ")";
                                    }
                                    list.add(column);
                                    seqsScan++;
                                } else {

                                    seqScan += "," + column + " tables";
                                    if (filter.equals("Filter")) {
                                        String filterContent = filters
                                                .getFirstChild()
                                                .getNodeValue();
                                        filterContent = filterContent
                                                .replace("::text", "");
                                        filterContent = filterContent
                                                .replace("(", "");
                                        filterContent = filterContent
                                                .replace(")", "");
                                        filterContent = filterContent
                                                .replace("&gt;=", "");
                                        filterContent = filterContent
                                                .replace("&gt;=", "");
                                        seqScan += " using filter ("
                                                + filterContent + ")";
                                    }
                                    list.add(column);
                                    seqsScan++;
```

```
                                }

                            }

                        }
                    }

                }

                if (root != 1) {// sub branch

                    if (element.getNodeName().equals("Node-Type")) {
                        if (element.getFirstChild().getNodeValue()
                                .equals("Bitmap Heap Scan")) {

                            noOfBitmapHeapScan++;
                            noOfBitmapHeapScanTables++;
                            Element tables = (Element) nodeList
                                    .item(i + 3);
                            String bhsTable = tables.getFirstChild()
                                    .getNodeValue();
                            if (noOfBitmapHeapScanTables == 1) {
                                bitmapTables += bhsTable;

                            } else
                                bitmapTables += "," + bhsTable;

                        }
                    }
                }

                // Bitmap index scan
```

```
if (root != 1) {// sub branch

    if (element.getNodeName().equals("Node-Type")) {
        if (element.getFirstChild().getNodeValue()
                .equals("Bitmap Index Scan")) {
            Element cond = (Element) nodeList
                    .item(i + 12);
            String acond = cond.getFirstChild()
                    .getNodeValue();
            acond = acond.replaceAll("::text", "");
            acond = acond.replace("(", "");
            acond = acond.replace(")", "");
            noOfBitmapIndexScan++;
            if (noOfBitmapIndexScan == 1)
                bitmapIndexScan += acond;
            else
                bitmapIndexScan += "," + acond;
        }
    }
}

// Index scan
```

```
if (root != 1) {// sub branch
    if (element.getNodeName().equals("Node-Type")) {
        if (element.getFirstChild().getNodeValue()
                .equals("Index Scan")) {
            noOfIndexScan++;
            Element tableNode = (Element) nodeList
                    .item(i + 5);
            Element tkey = (Element) nodeList
                    .item(i + 4);
            Element tfilter = (Element) nodeList
                    .item(i + 15);
            String table = tableNode.getFirstChild()
                    .getNodeValue();
            String key = tkey.getFirstChild()
                    .getNodeValue();
            String filter = tfilter.getFirstChild()
                    .getNodeValue();
            filter = filter.replaceAll("::text", "");
            filter = filter.replace("(", "");
            filter = filter.replace(")", "");
            if (noOfIndexScan == 1) {
                indexScan += "Index Scan(s) are used on "
                        + table
                        + " table using "
                        + key
                        + " key";
                if (tfilter.getNodeName() == "Filter") {

                    indexScan += " with " + filter
                            + " filter";
```

```
                    } else if (tfilter.getNodeName() == "Index-Cond") {
                        indexScan += " with " + filter
                                + " condition";

                        Element tfilter2 = (Element) nodeList
                                .item(i + 17);
                        if (tfilter2.getNodeName() == "Filter") {
                            String filter2 = tfilter2
                                    .getFirstChild()
                                    .getNodeValue();
                            filter2 = filter2.replaceAll(
                                    "::text", "");
                            filter2 = filter2.replace("(",
                                    "");
                            filter2 = filter2.replace(")",
                                    "");
                            indexScan += " and " + filter2;
                        }
                    }
                } else if (noOfIndexScan > 1) {
                    if (tfilter.getNodeName() == "Filter") {
                        indexScan += ", " + table
                                + " using " + key
                                + " key with " + filter
                                + " filter";
                    } else if (tfilter.getNodeName() == "Index-Cond") {
                        Element tfilter2 = (Element) nodeList
                                .item(i + 17);
                        if (tfilter2.getNodeName() == "Filter") {

                            String filter2 = tfilter2
                                    .getFirstChild()
                                     .getNodeValue();
                             filter2 = filter2.replaceAll(
                                     "::text", "");
                             filter2 = filter2.replace("(",
                                     "");
                             filter2 = filter2.replace(")",
                                     "");
                             indexScan += ", " + table
                                     + " using " + key
                                     + " key with " + filter
                                     + " filter and "
                                     + filter2
                                     + " condition";
                         } else {
                             indexScan += ", " + table
                                     + " using " + key
                                     + " key with " + filter
                                     + " condition";
                         }
                     } else
                         indexScan += ", " + table
                                 + " using " + key + " key";
                 }
             }
         }
     }

```

```
593                 if (root != 1) {// sub branch
594
595                     if (element.getNodeName().equals("Node-Type")) {
596                         if (element.getFirstChild().getNodeValue()
597                                 .equals("Index Only Scan")) {
598                             noOfIndexOnlyScan++;
599                             Element tableNode = (Element) nodeList
600                                     .item(i + 5);
601                             Element tkey = (Element) nodeList
602                                     .item(i + 4);
603                             Element tfilter = (Element) nodeList
604                                     .item(i + 15);
605                             String table = tableNode.getFirstChild()
606                                     .getNodeValue();
607                             String key = tkey.getFirstChild()
608                                     .getNodeValue();
609                             String filter = tfilter.getFirstChild()
610                                     .getNodeValue();
611                             filter = filter.replaceAll("::text", "");
612                             filter = filter.replace("(", "");
613                             filter = filter.replace(")", "");
614                             if (noOfIndexOnlyScan == 1) {
615                                 indexOnlyScan += "Index Only Scan(s) are used on "
616                                         + table
617                                         + " table using "
618                                         + key
619                                         + " key";
620                                 if (tfilter.getNodeName() == "Filter") {
621                                     indexOnlyScan += " with " + filter
622                                             + " filter";
623                                 } else if (tfilter.getNodeName() == "Index-Cond") {
624                                     indexOnlyScan += " with " + filter
625                                           + " condition";
626
627                                   Element tfilter2 = (Element) nodeList
628                                           .item(i + 17);
629                                   if (tfilter2.getNodeName() == "Filter") {
630                                       String filter2 = tfilter2
631                                               .getFirstChild()
632                                               .getNodeValue();
633                                       filter2 = filter2.replaceAll(
634                                               "::text", "");
635                                       filter2 = filter2.replace("(",
636                                               "");
637                                       filter2 = filter2.replace(")",
638                                               "");
639                                       indexScan += " and " + filter2;
640                                   }
641                               }
642                             } else if (noOfIndexOnlyScan > 1) {
643                                 if (tfilter.getNodeName() == "Filter") {
644                                     indexOnlyScan += ", " + table
645                                           + " using " + key
646                                           + " key with " + filter
647                                           + " filter";
648                                 } else if (tfilter.getNodeName() == "Index-Cond") {
649                                     Element tfilter2 = (Element) nodeList
650                                           .item(i + 17);
651                                     if (tfilter2.getNodeName() == "Filter") {
652
```

```
                                        String filter2 = tfilter2
                                                .getFirstChild()
                                                .getNodeValue();
                                        filter2 = filter2.replaceAll(
                                                "::text", "");
                                        filter2 = filter2.replace("(",
                                                "");
                                        filter2 = filter2.replace(")",
                                                "");
                                        indexOnlyScan += ", " + table
                                                + " using " + key
                                                + " key with " + filter
                                                + " filter and "
                                                + filter2
                                                + " condition";
                                    } else {
                                        indexOnlyScan += ", " + table
                                                + " using " + key
                                                + " key with " + filter
                                                + " condition";
                                    }
                                } else
                                    indexOnlyScan += ", " + table
                                            + " using " + key + " key";
                                }
                            }
                        }
                    }

                    // Merge Join
                    if (root != 1) {// sub branch

                        // TO DO skip repeated
                        boolean skip = false;

                        if (element.getNodeName().equals("Node-Type")) {
                            if (element.getFirstChild().getNodeValue()
                                    .equals("Merge Join")) {

                                Element mergeCond = (Element) nodeList
                                        .item(i + 11);

                                if (!mergeCond.getNodeName().equals(
                                        "Merge-Cond")) {
                                    mergeCond = (Element) nodeList
                                            .item(i + 12);
                                }

                                String mCond = mergeCond.getFirstChild()
                                        .getNodeValue();
                                mCond = mCond.replace("(", "");
                                mCond = mCond.replace(")", "");
                                mCond = mCond.replace("_", "");
                                mCond = mCond.replace("1", "");
                                mCond = mCond.replace("2", "");
                                mCond = mCond.replace("3", "");
                                mCond = mCond.replace("4", "");
                                mCond = mCond.replace("5", "");
                                mCond = mCond.replace("6", "");
                                mCond = mCond.replace("7", "");
```

```
                                mCond = mCond.replace("8", "");
                                for (int k = 0; k < mergelist.size(); k++) {
                                    if ((mCond.equals(mergelist.get(k)))) {
                                        skip = true;
                                    }
                                }
                                if (skip != true) {

                                    noOfMergeJoins++;
                                    if (noOfMergeJoins == 1)
                                        mergeJoins += "Merge Join(s) was used with conditons "
                                                + mCond;
                                    else {
                                        mergeJoins += "," + mCond;
                                    }

                                }
                                mergelist.add(mCond);
                            }

                        }
                    }

                }
                if (element.getNodeName().equals("Planning-Time")) {

                    planningTime = "The planning time of this query plan is "
                            + element.getFirstChild().getNodeValue()
                            + " milliseconds";

                }
                if (element.getNodeName().equals("Execution-Time")) {

                    exeTime = " and actual execution time of this query is "
                            + element.getFirstChild().getNodeValue()
                            + " milliseconds.";

                }

                // System.out.println(element.getNodeName() + ": " +
                // element.getFirstChild().getNodeValue());

            }
            String sortMethodPural = "The method(s) that were used for the Sort Operations are "
                    + sortMethod;
            String sortMethodSingluar = "The method that was used for the Sort Operation is "
                    + sortMethod;
            String bitmapHeapScanTables = "The tables that Bitmap Heap Scan Operation was performed on are "
                    + bitmapTables;
            String bitmapHeapScanTable = "The table that Bitmap Heap Scan Operation was performed on is "
                    + bitmapTables;
            if (hashsJoin > 0) {
                hashJoin += ".\n";
            }

```

```java

                if (noOfIndexOnlyScan > 0) {
                    indexOnlyScan += ".\n";
                }
                if (noOfIndexScan > 0) {
                    indexScan += ".\n";
                }
                if (seqsScan > 0) {
                    seqScan += ".\n";
                }
                String finalSortM = "";
                String finalBitMapTable = "";
                bitmapIndexScan += ".";
                if (noOfMergeJoins > 0) {
                    mergeJoins += ".\n";
                }
                if (noOfSort > 1) {
                    finalSortM = sortMethodPural + ".\n";
                } else if (noOfSort == 1)
                    finalSortM = sortMethodSingluar + ".\n";

                if (noOfBitmapHeapScanTables > 1) {
                    finalBitMapTable = bitmapHeapScanTables + ".\n";
                } else if (noOfBitmapHeapScanTables == 1)
                    finalBitMapTable = bitmapHeapScanTable + ".\n";
                if (noOfBitmapHeapScan > 1) {
                    bitmapHeapScan += noOfBitmapHeapScan
                            + " instances of Bitmap Heap Scan were performed during this Query.\n";
                }

                String bitmapIndexInstance = "";
                if (noOfBitmapIndexScan > 0) {
                    bitmapIndexInstance += "There are "
                            + noOfBitmapIndexScan
                            + " instance of Bitmap Index Scan performed during this Query using condition(s) "
                            + bitmapIndexScan + "\n";
                }
                String ttscOutput = output + hashJoin + seqScan + finalSortM
                        + bitmapHeapScan + finalBitMapTable
                        + bitmapIndexInstance + indexScan + indexOnlyScan
                        + mergeJoins + planningTime + exeTime;

                System.out.println(output + "\n" + hashJoin + seqScan
                        + finalSortM + bitmapHeapScan + finalBitMapTable
                        + bitmapIndexInstance + indexScan + indexOnlyScan
                        + mergeJoins + planningTime + exeTime);
                // ttsc.speak(ttscOutput);

            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                System.out.println("Invalid Query!");

            } catch (Exception e) {
                e.printStackTrace();
            }

        }
    }
```

2.3 Screenshots of Query Plan vs Text Description

Qn	Query Plan from PostgresSQL	Text Description (Natural Language)
1	**2.3.1 Question 1** QUERY PLAN text 1 HashAggregate (cost=1049261.90..1049261.95 rows=5 width=40) (actual time=10465.968..10465.968 rows=5 loops=1) 2 Group Key: ('Article'::text), (count(*)) 3 -> Append (cost=291752.48..1049261.88 rows=5 width=40) (actual time=4093.855..10465.938 rows=5 loops=1) 4 -> Aggregate (cost=291752.48..291752.50 rows=1 width=40) (actual time=4093.855..4093.855 rows=1 loops=1) 5 -> Hash Join (cost=60472.60..288161.25 rows=1436494 width=0) (actual time=659.765..4022.866 rows=1401146 loops=1) 6 Hash Cond: (publications."pubID" = articles."pubID") 7 -> Seq Scan on publications (cost=0.00..165103.14 rows=3247516 width=4) (actual time=0.026..1429.815 rows=3242071 loops=1) 8 Filter: ((year >= 2000) AND (year <= 2017)) 9 Rows Removed by Filter: 581578 10 -> Hash (cost=32706.82..32706.82 rows=1692382 width=4) (actual time=653.656..653.656 rows=1692382 loops=1) 11 Buckets: 131072 Batches: 32 Memory Usage: 2888kB 12 -> Seq Scan on articles (cost=0.00..32706.82 rows=1692382 width=4) (actual time=0.068..355.362 rows=1692382 loops=1) 13 -> Aggregate (cost=91546.15..91546.16 rows=1 width=40) (actual time=45.773..45.773 rows=1 loops=1) **Note: Screenshot only show a partial of the overall result. Result has a total of 46 rows shown below.** 'HashAggregate (cost=1049261.90..1049261.95 rows=5 width=40) (actual time=10465.968..10465.968 rows=5 loops=1)' ' Group Key: ('Article'::text), (count(*))' ' -> Append (cost=291752.48..1049261.88 rows=5 width=40) (actual time=4093.855..10465.938 rows=5 loops=1)' ' -> Aggregate (cost=291752.48..291752.50 rows=1 width=40) (actual time=4093.855..4093.855 rows=1 loops=1)' ' -> Hash Join (cost=60472.60..288161.25 rows=1436494 width=0) (actual time=659.765..4022.866 rows=1401146 loops=1)' ' Hash Cond: (publications."pubID" = articles."pubID")' ' -> Seq Scan on publications (cost=0.00..165103.14 rows=3247516 width=4) (actual time=0.026..1429.815 rows=3242071 loops=1)' ' Filter: ((year >= 2000) AND (year <= 2017))' ' Rows Removed by Filter: 581578' ' -> Hash (cost=32706.82..32706.82 rows=1692382 width=4) (actual time=653.656..653.656 rows=1692382 loops=1)' ' Buckets: 131072 Batches: 32 Memory Usage: 2888kB' ' -> Seq Scan on articles (cost=0.00..32706.82 rows=1692382 width=4) (actual time=0.068..355.362 rows=1692382 loops=1)' ' -> Aggregate (cost=91546.15..91546.16 rows=1 width=40) (actual time=45.773..45.773 rows=1 loops=1)' ' -> Nested Loop (cost=0.43..91516.98 rows=11668 width=0) (actual time=0.132..45.165 rows=8502 loops=1)' ' -> Seq Scan on books (cost=0.00..255.46 rows=13746 width=4) (actual time=0.060..2.954 rows=13746 loops=1)' ' -> Index Scan using publications_pkey on publications publications_1 (cost=0.43..6.63 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=13746)'	The execution of this Query begins by running an **Aggregate Operation** using **Hashed Strategy** on the result sets of its sub branches. In the Sub Branches, **Hash Join** Operations are performed between **publications and articles,publications and incollections,publications and inproceedings**. **Sequential Scan Operations** are also performed on **publications tables** using **filter (year >= 2000 AND year <= 2017),articles tables,books tables,incollections tables,proceedings tables,inproceedings tables**. **Index Scan(s)** are used on **publications table** using **publications_pkey key with "pubID" = books."pubID" condition and year >= 2000 AND year <= 2017**, **publications** using **publications_pkey key with "pubID" = proceedings."pubID" filter and year >= 2000 AND year <= 2017 condition**. The **planning time** of this query plan is **5.870 milliseconds** and **actual execution time** of this query is **8368.462 milliseconds**.

```
'                    Index Cond: ("pubID" = books."pubID")'
'                    Filter: ((year >= 2000) AND (year <= 2017))'
'                    Rows Removed by Filter: 0'
'      ->  Aggregate  (cost=179477.51..179477.52 rows=1 width=40) (actual time=1973.161..1973.161 rows=1 loops=1)'
'            ->  Hash Join  (cost=1714.18..179381.10 rows=38561 width=0) (actual time=844.606..1970.573 rows=43919 loops=1)'
'                  Hash Cond: (publications_2."pubID" = incollections."pubID")'
'                  ->  Seq Scan on publications publications_2  (cost=0.00..165103.14 rows=3247516 width=4) (actual time=0.019..1449.896 rows=3242071 loops=1)'
'                    Filter: ((year >= 2000) AND (year <= 2017))'
'                    Rows Removed by Filter: 581578'
'                  ->  Hash  (cost=1146.30..1146.30 rows=45430 width=4) (actual time=25.446..25.446 rows=45430 loops=1)'
'                    Buckets: 65536  Batches: 1  Memory Usage: 2110kB'
'                    ->  Seq Scan on incollections  (cost=0.00..1146.30 rows=45430 width=4) (actual time=0.047..17.940 rows=45430 loops=1)'
'      ->  Aggregate  (cost=178242.04..178242.05 rows=1 width=40) (actual time=325.981..325.981 rows=1 loops=1)'
'            ->  Nested Loop  (cost=0.43..178168.46 rows=29431 width=0) (actual time=0.140..323.537 rows=29456 loops=1)'
'                  ->  Seq Scan on proceedings  (cost=0.00..715.74 rows=34674 width=4) (actual time=0.076..6.471 rows=34674 loops=1)'
'                  ->  Index Scan using publications_pkey on publications publications_3  (cost=0.43..5.11 rows=1 width=4) (actual time=0.009..0.009 rows=1 loops=34674)'
'                    Index Cond: ("pubID" = proceedings."pubID")'
'                    Filter: ((year >= 2000) AND (year <= 2017))'
'                    Rows Removed by Filter: 0'
'      ->  Aggregate  (cost=308243.59..308243.60 rows=1 width=40) (actual time=4027.162..4027.163 rows=1 loops=1)'
'            ->  Hash Join  (cost=71954.88..303920.19 rows=1729359 width=0) (actual time=1189.525..3950.609 rows=1759048 loops=1)'
'                  Hash Cond: (publications_4."pubID" = inproceedings."pubID")'
'                  ->  Seq Scan on publications publications_4  (cost=0.00..165103.14 rows=3247516 width=4) (actual time=0.021..1336.943 rows=3242071 loops=1)'
'                    Filter: ((year >= 2000) AND (year <= 2017))'
'                    Rows Removed by Filter: 581578'
'                  ->  Hash  (cost=38528.17..38528.17 rows=2037417 width=4) (actual time=752.191..752.191 rows=2037417 loops=1)'
'                    Buckets: 131072  Batches: 32  Memory Usage: 3262kB'
'                    ->  Seq Scan on inproceedings  (cost=0.00..38528.17 rows=2037417 width=4) (actual time=0.032..400.586 rows=2037417 loops=1)'
'Planning time: 4.440 ms'
```

	'Execution time: 10467.146 ms'	
2	**2.3.2 Question 2** QUERY PLAN text 1 Append (cost=198185.25..204651.20 rows=73214 width=21) (actual time=1919.033..1971.457 rows=41 loops=1) 2 -> GroupAggregate (cost=198185.25..199805.00 rows=71989 width=21) (actual time=1919.032..1945.117 rows=41 loops=1) 3 Group Key: inproceedings.booktitle, publications.year 4 Filter: (count(*) > '200'::bigint) 5 Rows Removed by Filter: 12513 6 -> Sort (cost=198185.25..198365.22 rows=71989 width=13) (actual time=1917.515..1933.596 rows=74787 loops=1) 7 Sort Key: inproceedings.booktitle, publications.year 8 Sort Method: external merge Disk: 1792kB 9 -> Merge Join (cost=4.07..192377.36 rows=71989 width=13) (actual time=0.198..1534.405 rows=74787 loops=1) 10 Merge Cond: (inproceedings."inproceedingsID" = publications."pubID") 11 -> Index Scan using inproceedings_pkey on inproceedings (cost=0.43..71074.68 rows=2037417 width=13) (actual time=0.009..464.600 rows=2037417 loops=1) 12 -> Index Scan using publications_pkey on publications (cost=0.43..216619.86 rows=135186 width=8) (actual time=0.085..915.731 rows=74788 loops=1) 13 Filter: (month = 'July'::text) **Note: Screenshot only show a partial of the overall result. Result has a total of 26 rows shown below.** 'Append (cost=198185.25..204651.20 rows=73214 width=21) (actual time=1919.033..1971.457 rows=41 loops=1)' ' -> GroupAggregate (cost=198185.25..199805.00 rows=71989 width=21) (actual time=1919.032..1945.117 rows=41 loops=1)' ' Group Key: inproceedings.booktitle, publications.year' ' Filter: (count(*) > '200'::bigint)' ' Rows Removed by Filter: 12513' ' -> Sort (cost=198185.25..198365.22 rows=71989 width=13) (actual time=1917.515..1933.596 rows=74787 loops=1)' ' Sort Key: inproceedings.booktitle, publications.year' ' Sort Method: external merge Disk: 1792kB' ' -> Merge Join (cost=4.07..192377.36 rows=71989 width=13) (actual time=0.198..1534.405 rows=74787 loops=1)' ' Merge Cond: (inproceedings."inproceedingsID" = publications."pubID")' ' -> Index Scan using inproceedings_pkey on inproceedings (cost=0.43..71074.68 rows=2037417 width=13) (actual time=0.009..464.600 rows=2037417 loops=1)' ' -> Index Scan using publications_pkey on publications (cost=0.43..216619.86 rows=135186 width=8) (actual time=0.085..915.731 rows=74788 loops=1)' ' Filter: (month = 'July'::text)' ' Rows Removed by Filter: 1962641' ' -> HashAggregate (cost=3369.67..3381.92 rows=1225 width=23) (actual time=26.334..26.334 rows=0 loops=1)' ' Group Key: proceedings.booktitle, publications_1.year'	The execution of this Query begins by running an **Append Operation**. The method that was used for the **Sort Operation** is **external merge**. **Index Scan(s)** are used on **inproceedings table** using **inproceedings_pkey key, publications** using **publications_pkey key with month = 'July' filter, proceedings** using **"proceedingsID" key, publications** using **publications_pkey key with month = 'July' filter**. **Merge Join(s)** was used with **conditons inproceedings."inproceedingsID" = publications."pubID",proceedings."proceedingsID" = publications."pubID".** The **planning time** of this query plan is **3.449 milliseconds** and **actual execution time** of this query is **1958.982 milliseconds**.

<table>
<tr>
<td></td>
<td>
' Filter: (count(*) > '200'::bigint)'

' Rows Removed by Filter: 2093'

' -> Merge Join (cost=1.43..3357.42 rows=1225 width=15) (actual time=0.360..24.883 rows=3137 loops=1)'

' Merge Cond: (proceedings."proceedingsID" = publications_1."pubID")'

' -> Index Scan using "proceedingsID" on proceedings (cost=0.29..1282.45 rows=34674 width=15) (actual time=0.086..7.809 rows=34674 loops=1)'

' -> Index Scan using publications_pkey on publications publications_1 (cost=0.43..216619.86 rows=135186 width=8) (actual time=0.184..14.407 rows=3138 loops=1)'

' Filter: (month = 'July'::text)'

' Rows Removed by Filter: 32132'

'Planning time: 0.932 ms'

'Execution time: 1977.386 ms'
</td>
<td></td>
</tr>
<tr>
<td>3a</td>
<td>
2.3.3 Question 3a

<table>
<tr><th></th><th>QUERY PLAN
text</th></tr>
<tr><td>1</td><td>Nested Loop Left Join (cost=3.01..97.21 rows=1 width=378) (actual time=0.059..0.167 rows=7 loops=1)</td></tr>
<tr><td>2</td><td>-> Nested Loop Left Join (cost=2.72..96.88 rows=1 width=318) (actual time=0.052..0.142 rows=7 loops=1)</td></tr>
<tr><td>3</td><td>-> Nested Loop Left Join (cost=2.29..96.41 rows=1 width=282) (actual time=0.049..0.122 rows=7 loops=1)</td></tr>
<tr><td>4</td><td>-> Nested Loop Left Join (cost=2.00..96.09 rows=1 width=242) (actual time=0.046..0.113 rows=7 loops=1)</td></tr>
<tr><td>5</td><td>-> Nested Loop Left Join (cost=1.72..95.78 rows=1 width=209) (actual time=0.044..0.103 rows=7 loops=1)</td></tr>
<tr><td>6</td><td>-> Nested Loop (cost=1.29..95.31 rows=1 width=171) (actual time=0.040..0.089 rows=7 loops=1)</td></tr>
<tr><td>7</td><td>-> Nested Loop (cost=0.86..93.66 rows=3 width=8) (actual time=0.029..0.038 rows=16 loops=1)</td></tr>
<tr><td>8</td><td>-> Index Scan using authorsindex_authorname on authors (cost=0.43..8.45 rows=1 width=4) (actual time=0.020..0.021 rows=1 loops=1)</td></tr>
<tr><td>9</td><td>Index Cond: ("authorName" = 'Xiaokui Shu'::text)</td></tr>
<tr><td>10</td><td>-> Index Only Scan using authored_pkey on authored (cost=0.43..84.95 rows=26 width=8) (actual time=0.006..0.015 rows=16 loops=1)</td></tr>
<tr><td>11</td><td>Index Cond: ("authorID" = authors."authorID")</td></tr>
<tr><td>12</td><td>Heap Fetches: 16</td></tr>
<tr><td>13</td><td>-> Index Scan using publications_pkey on publications (cost=0.43..0.54 rows=1 width=167) (actual time=0.003..0.003 rows=0 loops=16)</td></tr>
</table>

Note: Screenshot only show a partial of the overall result. Result has a total of 28 rows shown below.

'Nested Loop Left Join (cost=3.01..97.21 rows=1 width=378) (actual time=0.063..0.176 rows=7 loops=1)'

' -> Nested Loop Left Join (cost=2.72..96.88 rows=1 width=318) (actual time=0.056..0.151 rows=7 loops=1)'

' -> Nested Loop Left Join (cost=2.29..96.41 rows=1 width=282) (actual time=0.053..0.131 rows=7 loops=1)'

' -> Nested Loop Left Join (cost=2.00..96.09 rows=1 width=242) (actual time=0.051..0.122 rows=7 loops=1)'

' -> Nested Loop Left Join (cost=1.72..95.78 rows=1 width=209) (actual time=0.049..0.114 rows=7 loops=1)'

' -> Nested Loop (cost=1.29..95.31 rows=1 width=171) (actual time=0.045..0.097 rows=7 loops=1)'

' -> Nested Loop (cost=0.86..93.66 rows=3 width=8) (actual time=0.035..0.045 rows=16 loops=1)'
</td>
<td>
The execution of this Query begins by running a Nested Loop Operation. Index Scan(s) are used on authors table using authorsindex_authorname key with "authorName" = 'Xiaokui Shu' condition, publications using publications_pkey key with "pubID" = authored."pubID" filter and year = '2015' condition, articles using articlesindex_pid key with authored."pubID" = "pubID" condition, books using booksindex_pubid key with authored."pubID" = "pubID" condition, incollections using incollections_pubid key with authored."pubID" = "pubID" condition, inproceedings using inproceedings_pubid key with authored."pubID" = "pubID" condition, proceedings using proceedings_pubid key with authored."pubID" = "pubID" condition.
Index Only Scan(s) are used on authored table using authored_pkey key with "authorID" = authors."authorID" condition.
The planning time of this query plan is 8.325 milliseconds and actual execution time of this query is 0.762 milliseconds.
</td>
</tr>
</table>

	' -> Index Scan using authorsindex_authorname on authors (cost=0.43..8.45 rows=1 width=4) (actual time=0.025..0.025 rows=1 loops=1)' ' Index Cond: ("authorName" = 'Xiaokui Shu'::text)' ' -> Index Only Scan using authored_pkey on authored (cost=0.43..84.95 rows=26 width=8) (actual time=0.007..0.016 rows=16 loops=1)' ' Index Cond: ("authorID" = authors."authorID")' ' Heap Fetches: 16' ' -> Index Scan using publications_pkey on publications (cost=0.43..0.54 rows=1 width=167) (actual time=0.003..0.003 rows=0 loops=16)' ' Index Cond: ("pubID" = authored."pubID")' ' Filter: (year = '2015'::text)' ' Rows Removed by Filter: 1' ' -> Index Scan using articlesindex_pid on articles (cost=0.43..0.46 rows=1 width=42) (actual time=0.002..0.002 rows=0 loops=7)' ' Index Cond: (authored."pubID" = "pubID")' ' -> Index Scan using booksindex_pubid on books (cost=0.29..0.30 rows=1 width=37) (actual time=0.001..0.001 rows=0 loops=7)' ' Index Cond: (authored."pubID" = "pubID")' ' -> Index Scan using incollections_pubid on incollections (cost=0.29..0.31 rows=1 width=44) (actual time=0.001..0.001 rows=0 loops=7)' ' Index Cond: (authored."pubID" = "pubID")' ' -> Index Scan using inproceedings_pubid on inproceedings (cost=0.43..0.46 rows=1 width=40) (actual time=0.002..0.003 rows=1 loops=7)' ' Index Cond: (authored."pubID" = "pubID")' ' -> Index Scan using proceedings_pubid on proceedings (cost=0.29..0.31 rows=1 width=54) (actual time=0.001..0.001 rows=0 loops=7)' ' Index Cond: (authored."pubID" = "pubID")' 'Planning time: 3.206 ms' 'Execution time: 0.280 ms'	
3b	**2.3.4 Question 3b**	The execution of this Query begins by running a **Nested Loop Operation**. In the Sub Branches, **Sequential Scan Operations** are performed on **authors tables** using **filter ("authorName" = 'Peter Nobel'),inproceedings tables** using **filter (booktitle = 'SIGMETRICS').**

	QUERY PLAN text
1	Nested Loop (cost=0.86..81345.44 rows=1 width=132) (actual time=438.415..1012.003 rows=1 loops=1)
2	Join Filter: (authored."pubID" = inproceedings."pubID")
3	Rows Removed by Join Filter: 2767
4	-> Nested Loop (cost=0.86..37720.16 rows=1 width=104) (actual time=50.816..332.141 rows=2 loops=1)
5	-> Nested Loop (cost=0.44..37716.71 rows=7 width=23) (actual time=50.760..331.927 rows=4 loops=1)
6	-> Seq Scan on authors (cost=0.00..37456.29 rows=1 width=19) (actual time=49.428..329.202 rows=1 loops=1)
7	Filter: ("authorName" = 'Peter Nobel'::text)
8	Rows Removed by Filter: 2000182
9	-> Index Only Scan using authored_pkey on authored (cost=0.44..259.75 rows=67 width=8) (actual time=1.324..2.712 rows=4 loops=1)
10	Index Cond: ("authorID" = authors."authorID")
11	Heap Fetches: 4
12	-> Index Scan using publications_pkey on publications (cost=0.43..0.48 rows=1 width=81) (actual time=0.048..0.050 rows=1 loops=4)
13	Index Cond: ("pubID" = authored."pubID")

Note: Screenshot only show a partial of the overall result. Result has a total of 20 rows shown below.

'Nested Loop (cost=0.86..81345.44 rows=1 width=132) (actual time=438.415..1012.003 rows=1 loops=1)'

' Join Filter: (authored."pubID" = inproceedings."pubID")'

' Rows Removed by Join Filter: 2767'

' -> Nested Loop (cost=0.86..37720.16 rows=1 width=104) (actual time=50.816..332.141 rows=2 loops=1)'

' -> Nested Loop (cost=0.44..37716.71 rows=7 width=23) (actual time=50.760..331.927 rows=4 loops=1)'

' -> Seq Scan on authors (cost=0.00..37456.29 rows=1 width=19) (actual time=49.428..329.202 rows=1 loops=1)'

' Filter: ("authorName" = 'Peter Nobel'::text)'

' Rows Removed by Filter: 2000182'

' -> Index Only Scan using authored_pkey on authored (cost=0.44..259.75 rows=67 width=8) (actual time=1.324..2.712 rows=4 loops=1)'

' Index Cond: ("authorID" = authors."authorID")'

' Heap Fetches: 4'

' -> Index Scan using publications_pkey on publications (cost=0.43..0.48 rows=1 width=81) (actual time=0.048..0.050 rows=1 loops=4)'

' Index Cond: ("pubID" = authored."pubID")'

' Filter: (year = 2010)'

' Rows Removed by Filter: 1'

' -> Seq Scan on inproceedings (cost=0.00..43621.71 rows=285 width=36) (actual time=46.119..339.832 rows=1384 loops=2)'

' Filter: (booktitle = 'SIGMETRICS'::text)'

' Rows Removed by Filter: 2036033'

'Planning time: 31.772 ms'

'Execution time: 1012.053 ms'

Index Scan(s) are used on **publications table** using **publications_pkey key with "pubID" = authored."pubID" condition and year = 2010.**
Index Only Scan(s) are used on **authored table** using **authored_pkey key with "authorID" = authors."authorID" condition.**
The **planning time** of this query plan is **5.978 milliseconds** and **actual execution time** of this query is **810.801 milliseconds**.

3c

2.3.5 Question 3c

	QUERY PLAN text
1	GroupAggregate (cost=809796.52..822611.37 rows=512594 width=100) (actual time=14426.585..15891.558 rows=643 loops=1)
2	Group Key: authors."authorName", publications.title, publications.year
3	Filter: (count(*) > 1)
4	Rows Removed by Filter: 525607
5	-> Sort (cost=809796.52..811078.01 rows=512594 width=92) (actual time=14422.074..15717.767 rows=526903 loops=1)
6	Sort Key: authors."authorName", publications.title
7	Sort Method: external merge Disk: 55352kB
8	-> Hash Join (cost=302265.40..708620.50 rows=512594 width=92) (actual time=4194.586..10385.900 rows=526903 loops=1)
9	Hash Cond: (authored."authorID" = authors."authorID")
10	-> Hash Join (cost=233087.28..606015.47 rows=512594 width=81) (actual time=3359.243..8060.744 rows=526903 loops=1)
11	Hash Cond: (authored."pubID" = publications."pubID")
12	-> Seq Scan on authored (cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.015..1630.973 rows=14079600 loops=1)
13	-> Hash (cost=229441.14..229441.14 rows=139291 width=85) (actual time=2514.408..2514.408 rows=140675 loops=1)

Note: Screenshot only show a partial of the overall result. Result has a total of 27 rows shown below.

'GroupAggregate (cost=809796.52..822611.37 rows=512594 width=100) (actual time=14426.585..15891.558 rows=643 loops=1)'

' Group Key: authors."authorName", publications.title, publications.year'

' Filter: (count(*) > 1)'

' Rows Removed by Filter: 525607'

' -> Sort (cost=809796.52..811078.01 rows=512594 width=92) (actual time=14422.074..15717.767 rows=526903 loops=1)'

' Sort Key: authors."authorName", publications.title'

' Sort Method: external merge Disk: 55352kB'

' -> Hash Join (cost=302265.40..708620.50 rows=512594 width=92) (actual time=4194.586..10385.900 rows=526903 loops=1)'

' Hash Cond: (authored."authorID" = authors."authorID")'

' -> Hash Join (cost=233087.28..606015.47 rows=512594 width=81) (actual time=3359.243..8060.744 rows=526903 loops=1)'

' Hash Cond: (authored."pubID" = publications."pubID")'

' -> Seq Scan on authored (cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.015..1630.973 rows=14079600 loops=1)'

' -> Hash (cost=229441.14..229441.14 rows=139291 width=85) (actual time=2514.408..2514.408 rows=140675 loops=1)'

' Buckets: 32768 Batches: 8 Memory Usage: 2348kB'

' -> Hash Join (cost=162384.75..229441.14 rows=139291 width=85) (actual time=1198.503..2460.421 rows=140675 loops=1)'

' Hash Cond: (inproceedings."pubID" = publications."pubID")'

' -> Seq Scan on inproceedings (cost=0.00..38528.17 rows=2037417 width=4) (actual time=0.033..383.654 rows=2037417 loops=1)'

The execution of this Query begins by running an **Aggregate Operation** using **Sorted Strategy** on the result sets of its sub branches. In the Sub Branches, **Hash Join Operations** are performed between **authored and authors,authored and publications,inproceedings and publications**.
Sequential Scan Operations are also performed on **authored tables,inproceedings tables,publications tables** using **filter (year = 2015),authors tables.**
The method that was used for the **Sort Operation** is **external merge**.
The **planning time** of this query plan is **2.871 milliseconds** and **actual execution time** of this query is **11879.690 milliseconds**.

<table>
<tr>
<td></td>
<td>' -> Hash (cost=155538.11..155538.11 rows=261571 width=81) (actual time=1193.286..1193.286 rows=269078 loops=1)'

' Buckets: 32768 Batches: 16 Memory Usage: 2223kB'

' -> Seq Scan on publications (cost=0.00..155538.11 rows=261571 width=81) (actual time=0.015..1071.648 rows=269078 loops=1)'

' Filter: (year = 2015)'

' Rows Removed by Filter: 3554571'

' -> Hash (cost=32455.83..32455.83 rows=2000183 width=19) (actual time=834.732..834.732 rows=2000183 loops=1)'

' Buckets: 65536 Batches: 32 Memory Usage: 3746kB'

' -> Seq Scan on authors (cost=0.00..32455.83 rows=2000183 width=19) (actual time=0.021..339.695 rows=2000183 loops=1)'

'Planning time: 0.884 ms'

'Execution time: 15912.687 ms'</td>
<td></td>
</tr>
<tr>
<td>4a</td>
<td>2.3.6 Question 4a

<table>
<tr><th></th><th>QUERY PLAN
text</th></tr>
<tr><td>1</td><td>Merge Join (cost=616508.26..620950.85 rows=1426 width=31) (actual time=58752.308..59118.686 rows=81 loops=1)</td></tr>
<tr><td>2</td><td>Merge Cond: (authors."authorName" = authors_2."authorName")</td></tr>
<tr><td>3</td><td>-> Nested Loop (cost=308254.34..312676.90 rows=534 width=38) (actual time=48910.093..49292.276 rows=271 loops=1)</td></tr>
<tr><td>4</td><td>-> GroupAggregate (cost=308253.91..308264.59 rows=534 width=23) (actual time=48870.871..48877.346 rows=271 loops=1)</td></tr>
<tr><td>5</td><td>Group Key: authors_1."authorName"</td></tr>
<tr><td>6</td><td>Filter: (count(*) >= '10'::bigint)</td></tr>
<tr><td>7</td><td>Rows Removed by Filter: 8126</td></tr>
<tr><td>8</td><td>-> Sort (cost=308253.91..308255.25 rows=534 width=15) (actual time=48870.807..48873.900 rows=18930 loops=1)</td></tr>
<tr><td>9</td><td>Sort Key: authors_1."authorName"</td></tr>
<tr><td>10</td><td>Sort Method: quicksort Memory: 1838kB</td></tr>
<tr><td>11</td><td>-> Nested Loop (cost=155264.81..308229.72 rows=534 width=15) (actual time=10436.535..48811.427 rows=18930 loops=1)</td></tr>
<tr><td>12</td><td>-> Hash Join (cost=155264.39..307979.03 rows=534 width=4) (actual time=10407.478..16729.491 rows=18930 loops=1)</td></tr>
<tr><td>13</td><td>Hash Cond: (authored."pubID" = publications."pubID")</td></tr>
</table>

'Merge Join (cost=616508.26..620950.85 rows=1426 width=31) (actual time=58752.308..59118.686 rows=81 loops=1)'

' Merge Cond: (authors."authorName" = authors_2."authorName")'

' -> Nested Loop (cost=308254.34..312676.90 rows=534 width=38) (actual time=48910.093..49292.276 rows=271 loops=1)'

' -> GroupAggregate (cost=308253.91..308264.59 rows=534 width=23) (actual time=48870.871..48877.346 rows=271 loops=1)'

' Group Key: authors_1."authorName"'

' Filter: (count(*) >= '10'::bigint)'

' Rows Removed by Filter: 8126'

' -> Sort (cost=308253.91..308255.25 rows=534 width=15) (actual time=48870.807..48873.900 rows=18930 loops=1)'

' Sort Key: authors_1."authorName"'

' Sort Method: quicksort Memory: 1838kB'</td>
<td>The execution of this Query begins by running a Merge Join Operation.In the Sub Branches, Hash Join Operations are performed between authored and publications,authored and publications. Sequential Scan Operations are also performed on authored tables,publications tables using filter ("pubKey" ~~ '%sigmod%').
The method(s) that were used for the Sort Operations are quicksort.
Index Scan(s) are used on authors table using authors_pkey key with "authorID" = authored."authorID" condition, authors using authors_pkey key with "authorID" = authored_1."authorID" condition.
Index Only Scan(s) are used on authors table using authorsindex_authorname key with "authorName" = authors_1."authorName" condition.
Merge Join(s) was used with conditons authors."authorName" = authors."authorName".
The planning time of this query plan is 2.984 milliseconds and actual execution time of this query is 7049.558 milliseconds.</td>
</tr>
</table>

	' -> Nested Loop (cost=155264.81..308229.72 rows=534 width=15) (actual time=10436.535..48811.427 rows=18930 loops=1)' ' -> Hash Join (cost=155264.39..307979.03 rows=534 width=4) (actual time=10407.478..16729.491 rows=18930 loops=1)' ' Hash Cond: (authored."pubID" = publications."pubID")' ' -> Seq Scan on authored (cost=0.00..132665.95 rows=5344895 width=8) (actual time=3662.172..9661.235 rows=14079600 loops=1)' ' -> Hash (cost=155259.61..155259.61 rows=382 width=4) (actual time=5794.247..5794.247 rows=6036 loops=1)' ' Buckets: 8192 (originally 1024) Batches: 1 (originally 1) Memory Usage: 277kB' ' -> Seq Scan on publications (cost=0.00..155259.61 rows=382 width=4) (actual time=395.102..5793.426 rows=6036 loops=1)' ' Filter: ("pubKey" ~~ '%sigmod%'::text)' ' Rows Removed by Filter: 3817613' ' -> Index Scan using authors_pkey on authors authors_1 (cost=0.43..0.46 rows=1 width=19) (actual time=1.692..1.694 rows=1 loops=18930)' ' Index Cond: ("authorID" = authored."authorID")' ' -> Index Only Scan using authorsindex_authorname on authors (cost=0.43..8.24 rows=1 width=15) (actual time=1.527..1.529 rows=1 loops=271)' ' Index Cond: ("authorName" = authors_1."authorName")' ' Heap Fetches: 271' ' -> Materialize (cost=308253.91..308271.27 rows=534 width=23) (actual time=9822.786..9825.736 rows=96 loops=1)' ' -> GroupAggregate (cost=308253.91..308264.59 rows=534 width=23) (actual time=9822.779..9825.664 rows=96 loops=1)' ' Group Key: authors_2."authorName"' ' Filter: (count(*) >= '10'::bigint)' ' Rows Removed by Filter: 4161' ' -> Sort (cost=308253.91..308255.25 rows=534 width=15) (actual time=9822.750..9823.930 rows=8347 loops=1)' ' Sort Key: authors_2."authorName"' ' Sort Method: quicksort Memory: 844kB' ' -> Nested Loop (cost=155264.81..308229.72 rows=534 width=15) (actual time=1928.869..9800.106 rows=8360 loops=1)'	

<table>
<tr>
<td></td>
<td>' -> Hash Join (cost=155264.39..307979.03 rows=534 width=4) (actual time=1928.855..3223.109 rows=8360 loops=1)'

' Hash Cond: (authored_1."pubID" = publications_1."pubID")'

' -> Seq Scan on authored authored_1 (cost=0.00..132665.95 rows=5344895 width=8) (actual time=60.940..1092.849 rows=14079600 loops=1)'

' -> Hash (cost=155259.61..155259.61 rows=382 width=4) (actual time=980.155..980.155 rows=1998 loops=1)'

' Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 87kB'

' -> Seq Scan on publications publications_1 (cost=0.00..155259.61 rows=382 width=4) (actual time=445.309..979.941 rows=1998 loops=1)'

' Filter: ("pubKey" ~~ '%pvldb%'::text)'

' Rows Removed by Filter: 3821651'

' -> Index Scan using authors_pkey on authors authors_2 (cost=0.43..0.46 rows=1 width=19) (actual time=0.786..0.786 rows=1 loops=8360)'

' Index Cond: ("authorID" = authored_1."authorID")'

'Planning time: 71.797 ms'

'Execution time: 59131.425 ms'</td>
<td></td>
</tr>
<tr>
<td>4b</td>
<td>2.3.7 Question 4b

<table>
<tr><th></th><th>QUERY PLAN
text</th></tr>
<tr><td>1</td><td>HashAggregate (cost=3837103.90..3837107.90 rows=400 width=40) (actual time=84177.284..84177.287 rows=37 loops=1)</td></tr>
<tr><td>2</td><td>Group Key: authors."authorName", (count(*))</td></tr>
<tr><td>3</td><td>-> Append (cost=2913766.32..3837101.90 rows=400 width=40) (actual time=75643.658..84177.255 rows=71 loops=1)</td></tr>
<tr><td>4</td><td>-> HashAggregate (cost=2913766.32..2913768.32 rows=200 width=23) (actual time=75643.656..75643.660 rows=37 loops=1)</td></tr>
<tr><td>5</td><td>Group Key: authors."authorName", (count(*))</td></tr>
<tr><td>6</td><td>-> Merge Join (cost=2574461.92..2843309.88 rows=14091289 width=23) (actual time=64342.626..75643.511 rows=37 loops=1)</td></tr>
<tr><td>7</td><td>Merge Cond: (authors."authorName" = authors_1."authorName")</td></tr>
<tr><td>8</td><td>-> GroupAggregate (cost=412176.21..412204.39 rows=1409 width=23) (actual time=4988.053..4990.947 rows=37 loops=1)</td></tr>
<tr><td>9</td><td>Group Key: authors."authorName"</td></tr>
<tr><td>10</td><td>Filter: (count(*) >= 15)</td></tr>
<tr><td>11</td><td>Rows Removed by Filter: 4230</td></tr>
<tr><td>12</td><td>-> Sort (cost=412176.21..412179.73 rows=1409 width=15) (actual time=4988.007..4989.329 rows=8360 loops=1)</td></tr>
<tr><td>13</td><td>Sort Key: authors."authorName"</td></tr>
</table>

Note: Screenshot only show a partial of the overall result. Result has a total of 95 rows shown below.

'HashAggregate (cost=3837103.90..3837107.90 rows=400 width=40) (actual time=84177.284..84177.287 rows=37 loops=1)'

' Group Key: authors."authorName", (count(*))'

' -> Append (cost=2913766.32..3837101.90 rows=400 width=40) (actual time=75643.658..84177.255 rows=71 loops=1)'</td>
<td>The execution of this Query begins by running an Aggregate Operation using Hashed Strategy on the result sets of its sub branches. In the Sub Branches, Hash Join Operations are performed between authored and publications,authored and authors,authored and publications,publications and inproceedings,authored and publications,authored and publications,publications and proceedings. Sequential Scan Operations are also performed on authored tables,publications tables using filter ("pubKey" ~~ '%pvldb%'),inproceedings tables using filter (booktitle !~~ '%KDD%'),authors tables,proceedings tables using filter (booktitle !~~ '%KDD%').
The method(s) that were used for the Sort Operations are quicksort,external merge. Index Scan(s) are used on authors table using authors_pkey key with "authorID" = authored."authorID" condition, authors</td>
</tr>
</table>

	' -> HashAggregate (cost=2913766.32..2913768.32 rows=200 width=23) (actual time=75643.656..75643.660 rows=37 loops=1)' ' Group Key: authors."authorName", (count(*))' ' -> Merge Join (cost=2574461.92..2843309.88 rows=14091289 width=23) (actual time=64342.626..75643.511 rows=37 loops=1)' ' Merge Cond: (authors."authorName" = authors_1."authorName")' ' -> GroupAggregate (cost=412176.21..412204.39 rows=1409 width=23) (actual time=4988.053..4990.947 rows=37 loops=1)' ' Group Key: authors."authorName"' ' Filter: (count(*) >= 15)' ' Rows Removed by Filter: 4230' ' -> Sort (cost=412176.21..412179.73 rows=1409 width=15) (actual time=4988.007..4989.329 rows=8360 loops=1)' ' Sort Key: authors."authorName"' ' Sort Method: quicksort Memory: 844kB' ' -> Nested Loop (cost=155543.33..412102.52 rows=1409 width=15) (actual time=2735.090..4954.341 rows=8360 loops=1)' ' -> Hash Join (cost=155542.90..411454.24 rows=1409 width=4) (actual time=2734.171..4659.436 rows=8360 loops=1)' ' Hash Cond: (authored."pubID" = publications."pubID")' ' -> Seq Scan on authored (cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.026..1563.005 rows=14079600 loops=1)' ' -> Hash (cost=155538.11..155538.11 rows=383 width=4) (actual time=1551.049..1551.049 rows=1998 loops=1)' ' Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 87kB' ' -> Seq Scan on publications (cost=0.00..155538.11 rows=383 width=4) (actual time=706.586..1550.730 rows=1998 loops=1)' ' Filter: ("pubKey" ~~ '%pvldb%'::text)' ' Rows Removed by Filter: 3821651' ' -> Index Scan using authors_pkey on authors (cost=0.43..0.45 rows=1 width=19) (actual time=0.035..0.035 rows=1 loops=8360)' ' Index Cond: ("authorID" = authored."authorID")' ' -> Materialize (cost=2162285.71..2224719.00 rows=2000183 width=15) (actual time=59206.707..70263.619 rows=1247262 loops=1)'	using **authors_pkey key with "authorID" = authored_2."authorID" condition, authors** using **authors_pkey key with "authorID" = authored_3."authorID" condition**. **Merge Join(s)** was used with **conditons authors."authorName" = authors."authorName"**. The **planning time** of this query plan is **74.710 milliseconds** and **actual execution time** of this query is **87347.294 milliseconds**.

	' -> Group (cost=2162285.71..2199716.71 rows=2000183 width=15) (actual time=59206.698..70125.036 rows=1247262 loops=1)' ' Group Key: authors_1."authorName"' ' -> Sort (cost=2162285.71..2181001.21 rows=7486200 width=15) (actual time=59206.696..69522.043 rows=6330905 loops=1)' ' Sort Key: authors_1."authorName"' ' Sort Method: external merge Disk: 158136kB' ' -> Hash Join (cost=402835.80..1051640.80 rows=7486200 width=15) (actual time=6726.566..17673.542 rows=6694980 loops=1)' ' Hash Cond: (authored_1."authorID" = authors_1."authorID")' ' -> Hash Join (cost=333657.68..799963.68 rows=7486200 width=4) (actual time=5852.838..12599.500 rows=6694980 loops=1)' ' Hash Cond: (authored_1."pubID" = publications_1."pubID")' ' -> Seq Scan on authored authored_1 (cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.018..1439.963 rows=14079600 loops=1)' ' -> Hash (cost=300282.17..300282.17 rows=2034281 width=8) (actual time=5605.653..5605.653 rows=2029079 loops=1)' ' Buckets: 131072 Batches: 32 Memory Usage: 3507kB' ' -> Hash Join (cost=76997.23..300282.17 rows=2034281 width=8) (actual time=1407.431..5195.925 rows=2029079 loops=1)' ' Hash Cond: (publications_1."pubID" = inproceedings."pubID")' ' -> Seq Scan on publications publications_1 (cost=0.00..145973.09 rows=3826009 width=4) (actual time=0.011..1662.625 rows=3823649 loops=1)' ' -> Hash (cost=43621.71..43621.71 rows=2034281 width=4) (actual time=1011.118..1011.118 rows=2029079 loops=1)' ' Buckets: 131072 Batches: 32 Memory Usage: 3252kB' ' -> Seq Scan on inproceedings (cost=0.00..43621.71 rows=2034281 width=4) (actual time=0.045..618.542 rows=2029079 loops=1)' ' Filter: (booktitle !~~ '%KDD%'::text)'	

```
'                                                    Rows Removed by Filter: 8338'

'                                  ->  Hash  (cost=32455.83..32455.83 rows=2000183
width=19) (actual time=872.583..872.583 rows=2000183 loops=1)'

'                                        Buckets: 65536  Batches: 32  Memory Usage:
3746kB'

'                                        ->  Seq Scan on authors authors_1
(cost=0.00..32455.83 rows=2000183 width=19) (actual time=0.030..360.549
rows=2000183 loops=1)'

'       ->  HashAggregate  (cost=923327.58..923329.58 rows=200 width=23)
(actual time=8533.590..8533.592 rows=34 loops=1)'

'             Group Key: authors_2."authorName", (count(*))'

'             ->  Merge Join  (cost=903390.50..918833.22 rows=898872 width=23)
(actual time=8488.814..8533.565 rows=34 loops=1)'

'                   Merge Cond: (authors_2."authorName" = authors_3."authorName")'

'                   ->  GroupAggregate  (cost=412176.21..412204.39 rows=1409
width=23) (actual time=3175.094..3177.458 rows=37 loops=1)'

'                         Group Key: authors_2."authorName"'

'                         Filter: (count(*) >= 15)'

'                         Rows Removed by Filter: 4230'

'                         ->  Sort  (cost=412176.21..412179.73 rows=1409 width=15)
(actual time=3175.062..3176.019 rows=8360 loops=1)'

'                               Sort Key: authors_2."authorName"'

'                               Sort Method: quicksort  Memory: 844kB'

'                               ->  Nested Loop  (cost=155543.33..412102.52 rows=1409
width=15) (actual time=1802.104..3153.562 rows=8360 loops=1)'

'                                     ->  Hash Join  (cost=155542.90..411454.24 rows=1409
width=4) (actual time=1802.089..3135.879 rows=8360 loops=1)'

'                                           Hash Cond: (authored_2."pubID" =
publications_2."pubID")'

'                                           ->  Seq Scan on authored authored_2
(cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.023..1036.951
rows=14079600 loops=1)'

'                                           ->  Hash  (cost=155538.11..155538.11 rows=383
width=4) (actual time=1071.279..1071.279 rows=1998 loops=1)'

'                                                 Buckets: 2048 (originally 1024)  Batches: 1
(originally 1)  Memory Usage: 87kB'

'                                                 ->  Seq Scan on publications publications_2
(cost=0.00..155538.11 rows=383 width=4) (actual time=471.295..1071.037
rows=1998 loops=1)'

'                                                       Filter: ("pubKey" ~~ '%pvldb%'::text)'
```

	' Rows Removed by Filter: 3821651' ' -> Index Scan using authors_pkey on authors authors_2 (cost=0.43..0.45 rows=1 width=19) (actual time=0.002..0.002 rows=1 loops=8360)' ' Index Cond: ("authorID" = authored_2."authorID")' ' -> Materialize (cost=491214.29..493447.11 rows=127590 width=15) (actual time=5313.293..5346.335 rows=32144 loops=1)' ' -> Group (cost=491214.29..491852.24 rows=127590 width=15) (actual time=5313.288..5342.980 rows=32144 loops=1)' ' Group Key: authors_3."authorName"' ' -> Sort (cost=491214.29..491533.26 rows=127590 width=15) (actual time=5313.286..5333.794 rows=86061 loops=1)' ' Sort Key: authors_3."authorName"' ' Sort Method: external merge Disk: 2104kB' ' -> Nested Loop (cost=162336.96..478213.42 rows=127590 width=15) (actual time=1252.213..4821.053 rows=88353 loops=1)' ' -> Hash Join (cost=162336.53..419509.68 rows=127590 width=4) (actual time=1252.196..3588.518 rows=88353 loops=1)' ' Hash Cond: (authored_3."pubID" = publications_3."pubID")' ' -> Seq Scan on authored authored_3 (cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.023..1205.823 rows=14079600 loops=1)' ' -> Hash (cost=161903.15..161903.15 rows=34671 width=8) (actual time=1229.078..1229.078 rows=34508 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1860kB' ' -> Hash Join (cost=1235.81..161903.15 rows=34671 width=8) (actual time=21.679..1223.029 rows=34508 loops=1)' ' Hash Cond: (publications_3."pubID" = proceedings."pubID")' ' -> Seq Scan on publications publications_3 (cost=0.00..145973.09 rows=3826009 width=4) (actual time=0.009..842.199 rows=3823649 loops=1)' ' -> Hash (cost=802.42..802.42 rows=34671 width=4) (actual time=10.545..10.545 rows=34508 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1726kB' ' -> Seq Scan on proceedings (cost=0.00..802.42 rows=34671 width=4) (actual time=0.011..7.000 rows=34508 loops=1)'	

<table>
<tr>
<td></td>
<td>' Filter: (booktitle !~~ '%KDD%'::text)'

' Rows Removed by Filter: 166'

' -> Index Scan using authors_pkey on authors authors_3 (cost=0.43..0.45 rows=1 width=19) (actual time=0.013..0.014 rows=1 loops=88353)'

' Index Cond: ("authorID" = authored_3."authorID")'

'Planning time: 3.055 ms'

'Execution time: 84212.445 ms'</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.3.8 Question 5

<table>
<tr><th></th><th>QUERY PLAN
text</th></tr>
<tr><td>1</td><td>Sort (cost=2349135.37..2349135.38 rows=5 width=40) (actual time=25071.136..25071.137 rows=5 loops=1)</td></tr>
<tr><td>2</td><td>Sort Key: ('1970-1979'::text)</td></tr>
<tr><td>3</td><td>Sort Method: quicksort Memory: 25kB</td></tr>
<tr><td>4</td><td>-> HashAggregate (cost=2349135.26..2349135.31 rows=5 width=40) (actual time=25071.121..25071.122 rows=5 loops=1)</td></tr>
<tr><td>5</td><td>Group Key: ('1970-1979'::text), (count(*))</td></tr>
<tr><td>6</td><td>-> Append (cost=379109.20..2349135.24 rows=5 width=40) (actual time=3943.243..25071.093 rows=5 loops=1)</td></tr>
<tr><td>7</td><td>-> Aggregate (cost=379109.20..379109.21 rows=1 width=40) (actual time=3943.241..3943.242 rows=1 loops=1)</td></tr>
<tr><td>8</td><td>-> HashAggregate (cost=378648.98..378853.52 rows=20454 width=8) (actual time=3939.441..3942.255 rows=12784 loops=1)</td></tr>
<tr><td>9</td><td>Group Key: publications."pubID", publications.year</td></tr>
<tr><td>10</td><td>-> Append (cost=1149.17..378546.71 rows=20454 width=8) (actual time=678.333..3933.866 rows=12784 loops=1)</td></tr>
<tr><td>11</td><td>-> Hash Join (cost=1149.17..166397.35 rows=342 width=8) (actual time=678.332..1553.145 rows=249 loops=1)</td></tr>
<tr><td>12</td><td>Hash Cond: (publications."pubID" = proceedings."pubID")</td></tr>
<tr><td>13</td><td>-> Seq Scan on publications (cost=0.00..165103.14 rows=37767 width=8) (actual time=0.023..1532.343 rows=39092 loops=1)</td></tr>
</table>

Note: Screenshot only show a partial of the overall result. Result has a total of 114 rows shown below.

'Sort (cost=2349135.37..2349135.38 rows=5 width=40) (actual time=21544.597..21544.598 rows=5 loops=1)'

' Sort Key: ('1970-1979'::text)'

' Sort Method: quicksort Memory: 25kB'

' -> HashAggregate (cost=2349135.26..2349135.31 rows=5 width=40) (actual time=21544.554..21544.555 rows=5 loops=1)'

' Group Key: ('1970-1979'::text), (count(*))'

' -> Append (cost=379109.20..2349135.24 rows=5 width=40) (actual time=3080.875..21544.518 rows=5 loops=1)'

' -> Aggregate (cost=379109.20..379109.21 rows=1 width=40) (actual time=3080.873..3080.873 rows=1 loops=1)'

' -> HashAggregate (cost=378648.98..378853.52 rows=20454 width=8) (actual time=3077.385..3080.006 rows=12784 loops=1)'

' Group Key: publications."pubID", publications.year'

' -> Append (cost=1149.17..378546.71 rows=20454 width=8) (actual time=17.233..3071.994 rows=12784 loops=1)'</td>
<td>The execution of this Query begins by running a Sort Operation.
In the Sub Branches, Hash Join Operations are performed between publications and proceedings,inproceedings and publications,publications and proceedings,inproceedings and publications,publications and proceedings,inproceedings and publications,publications and proceedings,inproceedings and publications,publications and proceedings,publications9 and inproceedings.
Sequential Scan Operations are also performed on publications tables using filter (year >= 1970 AND year <= 1979),proceedings tables,inproceedings tables.
The method(s) that were used for the Sort Operations are Memory,external merge.
The planning time of this query plan is 65.926 milliseconds and actual execution time of this query is 20035.179 milliseconds.</td>
</tr>
</table>

	' -> Hash Join (cost=1149.17..166397.35 rows=342 width=8) (actual time=17.232..1159.333 rows=249 loops=1)' ' Hash Cond: (publications."pubID" = proceedings."pubID")' ' -> Seq Scan on publications (cost=0.00..165103.14 rows=37767 width=8) (actual time=3.023..1139.187 rows=39092 loops=1)' ' Filter: ((year >= 1970) AND (year <= 1979))' ' Rows Removed by Filter: 3784557' ' -> Hash (cost=715.74..715.74 rows=34674 width=4) (actual time=13.240..13.240 rows=34674 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1732kB' ' -> Seq Scan on proceedings (cost=0.00..715.74 rows=34674 width=4) (actual time=0.012..8.150 rows=34674 loops=1)' ' -> Hash Join (cost=165575.22..211944.83 rows=20112 width=8) (actual time=1228.385..1912.020 rows=12535 loops=1)' ' Hash Cond: (inproceedings."pubID" = publications_1."pubID")' ' -> Seq Scan on inproceedings (cost=0.00..38528.17 rows=2037417 width=4) (actual time=0.026..354.113 rows=2037417 loops=1)' ' -> Hash (cost=165103.14..165103.14 rows=37767 width=8) (actual time=1214.841..1214.841 rows=39092 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 2040kB' ' -> Seq Scan on publications publications_1 (cost=0.00..165103.14 rows=37767 width=8) (actual time=3.442..1207.526 rows=39092 loops=1)' ' Filter: ((year >= 1970) AND (year <= 1979))' ' Rows Removed by Filter: 3784557' ' -> Aggregate (cost=398866.79..398866.80 rows=1 width=40) (actual time=4299.542..4299.542 rows=1 loops=1)' ' -> HashAggregate (cost=397539.27..398129.28 rows=59001 width=8) (actual time=4272.786..4295.792 rows=52263 loops=1)' ' Group Key: publications_2."pubID", publications_2.year' ' -> Append (cost=1149.17..397244.26 rows=59001 width=8) (actual time=14.031..4244.198 rows=52263 loops=1)' ' -> Hash Join (cost=1149.17..166670.71 rows=987 width=8) (actual time=14.030..1294.683 rows=1008 loops=1)' ' Hash Cond: (publications_2."pubID" = proceedings_1."pubID")'	

	' -> Seq Scan on publications publications_2 (cost=0.00..165103.14 rows=108943 width=8) (actual time=2.975..1262.948 rows=118472 loops=1)' ' Filter: ((year >= 1980) AND (year <= 1989))' ' Rows Removed by Filter: 3705177' ' -> Hash (cost=715.74..715.74 rows=34674 width=4) (actual time=10.233..10.233 rows=34674 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1732kB' ' -> Seq Scan on proceedings proceedings_1 (cost=0.00..715.74 rows=34674 width=4) (actual time=0.017..6.087 rows=34674 loops=1)' ' -> Hash Join (cost=166890.92..229983.55 rows=58014 width=8) (actual time=1456.554..2943.968 rows=51255 loops=1)' ' Hash Cond: (inproceedings_1."pubID" = publications_3."pubID")' ' -> Seq Scan on inproceedings inproceedings_1 (cost=0.00..38528.17 rows=2037417 width=4) (actual time=0.038..689.782 rows=2037417 loops=1)' ' -> Hash (cost=165103.14..165103.14 rows=108943 width=8) (actual time=1452.065..1452.065 rows=118472 loops=1)' ' Buckets: 131072 Batches: 2 Memory Usage: 3338kB' ' -> Seq Scan on publications publications_3 (cost=0.00..165103.14 rows=108943 width=8) (actual time=2.765..1417.147 rows=118472 loops=1)' ' Filter: ((year >= 1980) AND (year <= 1989))' ' Rows Removed by Filter: 3705177' ' -> Aggregate (cost=434333.46..434333.47 rows=1 width=40) (actual time=3902.870..3902.871 rows=1 loops=1)' ' -> Unique (cost=429955.80..431597.42 rows=218883 width=8) (actual time=3834.722..3893.856 rows=216667 loops=1)' ' -> Sort (cost=429955.80..430503.01 rows=218883 width=8) (actual time=3834.720..3866.154 rows=216667 loops=1)' ' Sort Key: publications_4."pubID", publications_4.year' ' Sort Method: external merge Disk: 3808kB' ' -> Append (cost=1149.17..407545.10 rows=218883 width=8) (actual time=13.250..3746.525 rows=216667 loops=1)' ' -> Hash Join (cost=1149.17..167804.51 rows=3663 width=8) (actual time=13.249..1350.138 rows=3921 loops=1)'	

	' Hash Cond: (publications_4."pubID" = proceedings_2."pubID")' ' -> Seq Scan on publications publications_4 (cost=0.00..165103.14 rows=404155 width=8) (actual time=1.285..1271.071 rows=411177 loops=1)' ' Filter: ((year >= 1990) AND (year <= 1999))' ' Rows Removed by Filter: 3412472' ' -> Hash (cost=715.74..715.74 rows=34674 width=4) (actual time=11.766..11.766 rows=34674 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1732kB' ' -> Seq Scan on proceedings proceedings_2 (cost=0.00..715.74 rows=34674 width=4) (actual time=0.017..7.186 rows=34674 loops=1)' ' -> Hash Join (cost=171734.07..237551.76 rows=215220 width=8) (actual time=1386.949..2389.485 rows=212746 loops=1)' ' Hash Cond: (inproceedings_2."pubID" = publications_5."pubID")' ' -> Seq Scan on inproceedings inproceedings_2 (cost=0.00..38528.17 rows=2037417 width=4) (actual time=0.040..348.672 rows=2037417 loops=1)' ' -> Hash (cost=165103.14..165103.14 rows=404155 width=8) (actual time=1386.086..1386.086 rows=411177 loops=1)' ' Buckets: 131072 Batches: 8 Memory Usage: 3037kB' ' -> Seq Scan on publications publications_5 (cost=0.00..165103.14 rows=404155 width=8) (actual time=0.819..1290.174 rows=411177 loops=1)' ' Filter: ((year >= 1990) AND (year <= 1999))' ' Rows Removed by Filter: 3412472' ' -> Aggregate (cost=535817.59..535817.60 rows=1 width=40) (actual time=5036.083..5036.083 rows=1 loops=1)' ' -> Unique (cost=521250.79..526713.34 rows=728340 width=8) (actual time=4716.194..5002.174 rows=784628 loops=1)' ' -> Sort (cost=521250.79..523071.64 rows=728340 width=8) (actual time=4716.192..4897.072 rows=784628 loops=1)' ' Sort Key: publications_6."pubID", publications_6.year' ' Sort Method: external merge Disk: 13792kB' ' -> Append (cost=1149.17..440370.40 rows=728340 width=8) (actual time=8.249..4437.416 rows=784628 loops=1)'	

	' -> Hash Join (cost=1149.17..171417.34 rows=12188 width=8) (actual time=8.249..1398.457 rows=12304 loops=1)' ' Hash Cond: (publications_6."pubID" = proceedings_3."pubID")' ' -> Seq Scan on publications publications_6 (cost=0.00..165103.14 rows=1344842 width=8) (actual time=0.094..1215.514 rows=1340513 loops=1)' ' Filter: ((year >= 2000) AND (year <= 2009))' ' Rows Removed by Filter: 2483136' ' -> Hash (cost=715.74..715.74 rows=34674 width=4) (actual time=7.762..7.762 rows=34674 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1732kB' ' -> Seq Scan on proceedings proceedings_3 (cost=0.00..715.74 rows=34674 width=4) (actual time=0.019..4.422 rows=34674 loops=1)' ' -> Hash Join (cost=187167.66..261669.66 rows=716152 width=8) (actual time=1531.993..3016.397 rows=772324 loops=1)' ' Hash Cond: (inproceedings_3."pubID" = publications_7."pubID")' ' -> Seq Scan on inproceedings inproceedings_3 (cost=0.00..38528.17 rows=2037417 width=4) (actual time=0.036..345.291 rows=2037417 loops=1)' ' -> Hash (cost=165103.14..165103.14 rows=1344842 width=8) (actual time=1520.577..1520.577 rows=1340513 loops=1)' ' Buckets: 131072 Batches: 32 Memory Usage: 2664kB' ' -> Seq Scan on publications publications_7 (cost=0.00..165103.14 rows=1344842 width=8) (actual time=0.043..1248.369 rows=1340513 loops=1)' ' Filter: ((year >= 2000) AND (year <= 2009))' ' Rows Removed by Filter: 2483136' ' -> Aggregate (cost=601008.10..601008.11 rows=1 width=40) (actual time=5225.142..5225.143 rows=1 loops=1)' ' -> Unique (cost=580399.10..588127.48 rows=1030450 width=8) (actual time=4825.770..5182.427 rows=1003885 loops=1)' ' -> Sort (cost=580399.10..582975.23 rows=1030450 width=8) (actual time=4825.769..5048.929 rows=1003885 loops=1)' ' Sort Key: publications_8."pubID", publications_8.year' ' Sort Method: external merge Disk: 17640kB'	

	' -> Append (cost=1149.17..463392.72 rows=1030450 width=8) (actual time=7.851..4422.549 rows=1003885 loops=1)' ' -> Hash Join (cost=1149.17..173559.76 rows=17243 width=8) (actual time=7.850..1350.693 rows=17161 loops=1)' ' Hash Cond: (publications_8."pubID" = proceedings_4."pubID")' ' -> Seq Scan on publications publications_8 (cost=0.00..165103.14 rows=1902674 width=8) (actual time=0.068..1131.857 rows=1901741 loops=1)' ' Filter: ((year >= 2010) AND (year <= 2019))' ' Rows Removed by Filter: 1921908' ' -> Hash (cost=715.74..715.74 rows=34674 width=4) (actual time=7.439..7.439 rows=34674 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1732kB' ' -> Seq Scan on proceedings proceedings_4 (cost=0.00..715.74 rows=34674 width=4) (actual time=0.009..4.236 rows=34674 loops=1)' ' -> Hash Join (cost=71954.88..279528.46 rows=1013207 width=8) (actual time=647.825..3043.178 rows=986724 loops=1)' ' Hash Cond: (publications_9."pubID" = inproceedings_4."pubID")' ' -> Seq Scan on publications publications_9 (cost=0.00..165103.14 rows=1902674 width=8) (actual time=0.091..1116.873 rows=1901741 loops=1)' ' Filter: ((year >= 2010) AND (year <= 2019))' ' Rows Removed by Filter: 1921908' ' -> Hash (cost=38528.17..38528.17 rows=2037417 width=4) (actual time=646.628..646.628 rows=2037417 loops=1)' ' Buckets: 131072 Batches: 32 Memory Usage: 3262kB' ' -> Seq Scan on inproceedings inproceedings_4 (cost=0.00..38528.17 rows=2037417 width=4) (actual time=0.021..340.925 rows=2037417 loops=1)' 'Planning time: 1.206 ms' 'Execution time: 21574.356 ms'	

6

2.3.9 Question 6

	QUERY PLAN text
1	Nested Loop (cost=1312850.51..1317642.64 rows=583 width=15) (actual time=78059.657..78059.658 rows=1 loops=1)
2	InitPlan 1 (returns $4)
3	-> Aggregate (cost=656428.68..656428.69 rows=1 width=8) (actual time=18983.395..18983.395 rows=1 loops=1)
4	-> Sort (cost=656419.94..656421.39 rows=583 width=12) (actual time=18976.198..18980.234 rows=51453 loops=1)
5	Sort Key: (count(DISTINCT authors_4."authorID")) DESC
6	Sort Method: quicksort Memory: 3778kB
7	-> GroupAggregate (cost=656382.95..656393.15 rows=583 width=12) (actual time=18764.849..18967.308 rows=51453 loops=1)
8	Group Key: authors_3."authorID"
9	-> Sort (cost=656382.95..656384.41 rows=583 width=8) (actual time=18764.819..18830.433 rows=336652 loops=1)
10	Sort Key: authors_3."authorID"
11	Sort Method: external merge Disk: 5912kB
12	-> Nested Loop (cost=503366.31..656356.17 rows=583 width=8) (actual time=13253.524..18566.882 rows=336652 loops=1)
13	Join Filter: (authors_3."authorID" <> authors_4."authorID")

Note: Screenshot only show a partial of the overall result. Result has a total of 137 rows.

'Nested Loop (cost=1312850.51..1317642.64 rows=583 width=15) (actual time=34323.323..34323.324 rows=1 loops=1)'

' InitPlan 1 (returns $4)'

' -> Aggregate (cost=656428.68..656428.69 rows=1 width=8) (actual time=16064.789..16064.789 rows=1 loops=1)'

' -> Sort (cost=656419.94..656421.39 rows=583 width=12) (actual time=16057.697..16061.857 rows=51453 loops=1)'

' Sort Key: (count(DISTINCT authors_4."authorID")) DESC'

' Sort Method: quicksort Memory: 3778kB'

' -> GroupAggregate (cost=656382.95..656393.15 rows=583 width=12) (actual time=15878.523..16049.872 rows=51453 loops=1)'

' Group Key: authors_3."authorID"'

' -> Sort (cost=656382.95..656384.41 rows=583 width=8) (actual time=15878.482..15932.493 rows=336652 loops=1)'

' Sort Key: authors_3."authorID"'

' Sort Method: external merge Disk: 5912kB'

' -> Nested Loop (cost=503366.31..656356.17 rows=583 width=8) (actual time=11276.959..15712.764 rows=336652 loops=1)'

' Join Filter: (authors_3."authorID" <> authors_4."authorID")'

' Rows Removed by Join Filter: 88188'

' -> Hash Join (cost=503365.89..656081.02 rows=583 width=8) (actual time=11276.931..14909.418 rows=424840 loops=1)'

' Hash Cond: (authored_3."pubID" = publications_2."pubID")'

The execution of this Query begins by running a **Nested Loop Operation**.In the Sub Branches, **Hash Join Operations** are performed between **authored and publications,authored** and **publications,proceedings and publications,authored** and **publications,authored and publications,proceedings** and **publications. Sequential Scan Operations** are also performed on **authored tables,proceedings tables** using filter **(booktitle ~~* '%DATA%'),inproceedings tables** using **filter (booktitle ~~* '%DATA%'),articles tables** using filter **(journal ~~* '%DATA%')**.

The method(s) that were used for the **Sort Operations** are **quicksort,external merge**. **Index Scan(s)** are used on **authors table** using **authors_pkey key** with **"authorID" = authors_1."authorID"** condition. **Index Only Scan(s)** are used on **publications table** using **publications_pkey key** with **"pubID" = proceedings_3."pubID" condition**, **publications** using **publications_pkey key** with **"pubID" = publications_2."pubID" condition**, **authors** using **authors_pkey key** with **"authorID" = authored_2."authorID" condition**, **authors** using **authors_pkey key** with **"authorID" = authored_3."authorID" condition**, **publications** using **publications_pkey key** with **"pubID" = proceedings_1."pubID" condition**, **publications** using **publications_pkey key** with **"pubID" = publications."pubID" condition**, **authors** using **authors_pkey key** with **"authorID" = authored."authorID" condition**, **authors** using **authors_pkey key** with **"authorID" = authored_1."authorID" condition**.

The **planning time** of this query plan is **41.063 milliseconds** and **actual execution time** of this query is **30967.372 milliseconds**.

	' -> Seq Scan on authored authored_3 (cost=0.00..132665.95 rows=5344895 width=8) (actual time=70.238..1249.181 rows=14079600 loops=1)' ' -> Hash (cost=503360.67..503360.67 rows=417 width=24) (actual time=11206.370..11206.370 rows=88188 loops=1)' ' Buckets: 65536 (originally 1024) Batches: 2 (originally 1) Memory Usage: 3585kB' ' -> Nested Loop (cost=350451.86..503360.67 rows=417 width=24) (actual time=8401.457..11183.087 rows=88188 loops=1)' ' -> Hash Join (cost=350451.43..503164.91 rows=417 width=24) (actual time=8401.449..10927.388 rows=88188 loops=1)' ' Hash Cond: (authored_2."pubID" = publications_2."pubID")' ' -> Seq Scan on authored authored_2 (cost=0.00..132665.95 rows=5344895 width=8) (actual time=0.014..1214.156 rows=14079600 loops=1)' ' -> Hash (cost=350447.71..350447.71 rows=298 width=16) (actual time=8401.423..8401.423 rows=28992 loops=1)' ' Buckets: 32768 (originally 1024) Batches: 1 (originally 1) Memory Usage: 1615kB' ' -> Nested Loop (cost=349469.17..350447.71 rows=298 width=16) (actual time=8300.347..8395.710 rows=28992 loops=1)' ' -> Hash Join (cost=349468.74..350273.79 rows=298 width=12) (actual time=8300.342..8318.417 rows=28992 loops=1)' ' Hash Cond: (proceedings_2."pubID" = publications_2."pubID")' ' -> HashAggregate (cost=81868.48..82206.19 rows=33771 width=36) (actual time=4091.635..4099.930 rows=28814 loops=1)' ' Group Key: proceedings_2."pubID", proceedings_2.booktitle' ' -> Append (cost=0.00..81699.62 rows=33771 width=36) (actual time=0.102..4080.419 rows=28814 loops=1)' ' -> Seq Scan on proceedings proceedings_2 (cost=0.00..802.42 rows=37 width=15) (actual time=0.102..33.627 rows=298 loops=1)' ' Filter: (booktitle ~~* '%DATA%'::text)' ' Rows Removed by Filter: 34376'	

	' -> Seq Scan on inproceedings inproceedings_2 (cost=0.00..43621.71 rows=148 width=13) (actual time=0.028..1864.763 rows=9015 loops=1)' ' Filter: (booktitle ~~* '%DATA%'::text)' ' Rows Removed by Filter: 2028402' ' -> Seq Scan on articles articles_2 (cost=0.00..36937.78 rows=33586 width=26) (actual time=13.081..2181.059 rows=19501 loops=1)' ' Filter: (journal ~~* '%DATA%'::text)' ' Rows Removed by Filter: 1672881' ' -> Hash (cost=267178.13..267178.13 rows=33771 width=8) (actual time=4208.315..4208.315 rows=28814 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1638kB' ' -> Nested Loop (cost=81868.91..267178.13 rows=33771 width=8) (actual time=4115.635..4204.064 rows=28814 loops=1)' ' -> HashAggregate (cost=81868.48..82206.19 rows=33771 width=36) (actual time=4115.555..4122.569 rows=28814 loops=1)' ' Group Key: proceedings_3."pubID", proceedings_3.booktitle' ' -> Append (cost=0.00..81699.62 rows=33771 width=36) (actual time=0.154..4103.360 rows=28814 loops=1)' ' -> Seq Scan on proceedings proceedings_3 (cost=0.00..802.42 rows=37 width=15) (actual time=0.153..38.905 rows=298 loops=1)' ' Filter: (booktitle ~~* '%DATA%'::text)' ' Rows Removed by Filter: 34376' ' -> Seq Scan on inproceedings inproceedings_3 (cost=0.00..43621.71 rows=148 width=13) (actual time=0.038..1905.983 rows=9015 loops=1)' ' Filter: (booktitle ~~* '%DATA%'::text)'	

```
'                                                                    Rows
Removed by Filter: 2028402'

'                                                              ->  Seq Scan on
articles articles_3  (cost=0.00..36937.78 rows=33586 width=26) (actual
time=10.975..2157.195 rows=19501 loops=1)'

'                                                                    Filter: (journal
~~* '%DATA%'::text)'

'                                                                    Rows
Removed by Filter: 1672881'

'                                                         ->  Index Only Scan using
publications_pkey on publications publications_2  (cost=0.43..5.46 rows=1
width=4) (actual time=0.002..0.003 rows=1 loops=28814)'

'                                                              Index Cond: ("pubID"
= proceedings_3."pubID")'

'                                                              Heap Fetches: 28814'

'                                                   ->  Index Only Scan using
publications_pkey on publications publications_3  (cost=0.43..0.57 rows=1
width=4) (actual time=0.002..0.002 rows=1 loops=28992)'

'                                                        Index Cond: ("pubID" =
publications_2."pubID")'

'                                                        Heap Fetches: 28992'

'                                         ->  Index Only Scan using authors_pkey on
authors authors_3  (cost=0.43..0.46 rows=1 width=4) (actual time=0.003..0.003
rows=1 loops=88188)'

'                                              Index Cond: ("authorID" =
authored_2."authorID")'

'                                              Heap Fetches: 88188'

'                                 ->  Index Only Scan using authors_pkey on authors
authors_4  (cost=0.43..0.46 rows=1 width=4) (actual time=0.002..0.002 rows=1
loops=424840)'

'                                    Index Cond: ("authorID" = authored_3."authorID")'

'                                    Heap Fetches: 424840'

'  ->  Sort  (cost=656421.39..656422.85 rows=583 width=12) (actual
time=34323.294..34323.294 rows=1 loops=1)'

'        Sort Key: (count(DISTINCT authors_2."authorID")) DESC'

'        Sort Method: quicksort  Memory: 25kB'

'        ->  GroupAggregate  (cost=656382.95..656394.61 rows=583 width=12)
(actual time=34114.119..34323.285 rows=1 loops=1)'

'              Group Key: authors_1."authorID"'

'              Filter: (count(DISTINCT authors_2."authorID") = $4)'
```

```
'            Rows Removed by Filter: 51452'

'            ->  Sort  (cost=656382.95..656384.41 rows=583 width=8) (actual time=18047.929..18119.898 rows=336652 loops=1)'

'                  Sort Key: authors_1."authorID"'

'                  Sort Method: external merge  Disk: 5912kB'

'                  ->  Nested Loop  (cost=503366.31..656356.17 rows=583 width=8) (actual time=13370.293..17878.571 rows=336652 loops=1)'

'                        Join Filter: (authors_1."authorID" <> authors_2."authorID")'

'                        Rows Removed by Join Filter: 88188'

'                        ->  Hash Join  (cost=503365.89..656081.02 rows=583 width=8) (actual time=13370.250..17039.535 rows=424840 loops=1)'

'                              Hash Cond: (authored_1."pubID" = publications."pubID")'

'                              ->  Seq Scan on authored authored_1  (cost=0.00..132665.95 rows=5344895 width=8) (actual time=83.559..1282.644 rows=14079600 loops=1)'

'                              ->  Hash  (cost=503360.67..503360.67 rows=417 width=24) (actual time=13285.982..13285.982 rows=88188 loops=1)'

'                                    Buckets: 65536 (originally 1024)  Batches: 2 (originally 1)  Memory Usage: 3585kB'

'                                    ->  Nested Loop  (cost=350451.86..503360.67 rows=417 width=24) (actual time=10423.921..13262.662 rows=88188 loops=1)'

'                                          ->  Hash Join  (cost=350451.43..503164.91 rows=417 width=24) (actual time=10423.912..13007.877 rows=88188 loops=1)'

'                                                Hash Cond: (authored."pubID" = publications."pubID")'

'                                                ->  Seq Scan on authored  (cost=0.00..132665.95 rows=5344895 width=8) (actual time=0.030..1256.244 rows=14079600 loops=1)'

'                                                ->  Hash  (cost=350447.71..350447.71 rows=298 width=16) (actual time=10423.839..10423.839 rows=28992 loops=1)'

'                                                      Buckets: 32768 (originally 1024)  Batches: 1 (originally 1)  Memory Usage: 1615kB'

'                                                      ->  Nested Loop  (cost=349469.17..350447.71 rows=298 width=16) (actual time=10326.633..10418.404 rows=28992 loops=1)'

'                                                            ->  Hash Join (cost=349468.74..350273.79 rows=298 width=12) (actual time=10326.628..10343.551 rows=28992 loops=1)'

'                                                                  Hash Cond: (proceedings."pubID" = publications."pubID")'
```

	' -> HashAggregate (cost=81868.48..82206.19 rows=33771 width=36) (actual time=5248.428..5256.319 rows=28814 loops=1)' ' Group Key: proceedings."pubID", proceedings.booktitle' ' -> Append (cost=0.00..81699.62 rows=33771 width=36) (actual time=0.071..5233.638 rows=28814 loops=1)' ' -> Seq Scan on proceedings (cost=0.00..802.42 rows=37 width=15) (actual time=0.071..36.631 rows=298 loops=1)' ' Filter: (booktitle ~~* '%DATA%'::text)' ' Rows Removed by Filter: 34376' ' -> Seq Scan on inproceedings (cost=0.00..43621.71 rows=148 width=13) (actual time=0.026..2318.777 rows=9015 loops=1)' ' Filter: (booktitle ~~* '%DATA%'::text)' ' Rows Removed by Filter: 2028402' ' -> Seq Scan on articles (cost=0.00..36937.78 rows=33586 width=26) (actual time=25.688..2876.521 rows=19501 loops=1)' ' Filter: (journal ~~* '%DATA%'::text)' ' Rows Removed by Filter: 1672881' ' -> Hash (cost=267178.13..267178.13 rows=33771 width=8) (actual time=5078.021..5078.021 rows=28814 loops=1)' ' Buckets: 65536 Batches: 1 Memory Usage: 1638kB' ' -> Nested Loop (cost=81868.91..267178.13 rows=33771 width=8) (actual time=4972.622..5073.431 rows=28814 loops=1)' ' -> HashAggregate (cost=81868.48..82206.19 rows=33771 width=36) (actual time=4972.578..4980.629 rows=28814 loops=1)' ' Group Key: proceedings_1."pubID", proceedings_1.booktitle' ' -> Append (cost=0.00..81699.62 rows=33771 width=36) (actual time=0.052..4957.168 rows=28814 loops=1)'	

	' -> Seq Scan on proceedings proceedings_1 (cost=0.00..802.42 rows=37 width=15) (actual time=0.051..33.584 rows=298 loops=1)' ' Filter: (booktitle ~~* '%DATA%'::text)' ' Rows Removed by Filter: 34376' ' -> Seq Scan on inproceedings inproceedings_1 (cost=0.00..43621.71 rows=148 width=13) (actual time=0.113..2171.397 rows=9015 loops=1)' ' Filter: (booktitle ~~* '%DATA%'::text)' ' Rows Removed by Filter: 2028402' ' -> Seq Scan on articles articles_1 (cost=0.00..36937.78 rows=33586 width=26) (actual time=10.564..2750.796 rows=19501 loops=1)' ' Filter: (journal ~~* '%DATA%'::text)' ' Rows Removed by Filter: 1672881' ' -> Index Only Scan using publications_pkey on publications (cost=0.43..5.46 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=28814)' ' Index Cond: ("pubID" = proceedings_1."pubID")' ' Heap Fetches: 28814' ' -> Index Only Scan using publications_pkey on publications publications_1 (cost=0.43..0.57 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=28992)' ' Index Cond: ("pubID" = publications."pubID")' ' Heap Fetches: 28992' ' -> Index Only Scan using authors_pkey on authors authors_1 (cost=0.43..0.46 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=88188)' ' Index Cond: ("authorID" = authored."authorID")' ' Heap Fetches: 88188' ' -> Index Only Scan using authors_pkey on authors authors_2 (cost=0.43..0.46 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=424840)' ' Index Cond: ("authorID" = authored_1."authorID")'	

<table>
<tr>
<td></td>
<td>' Heap Fetches: 424840'

' -> Index Scan using authors_pkey on authors (cost=0.43..8.20 rows=1 width=19) (actual time=0.025..0.025 rows=1 loops=1)'

' Index Cond: ("authorID" = authors_1."authorID")'

'Planning time: 38.754 ms'

'Execution time: 34335.507 ms'</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2.3.10 Question 7

<table>
<tr><th></th><th>QUERY PLAN
text</th></tr>
<tr><td>1</td><td>Limit (cost=3522534.11..3522534.13 rows=10 width=27) (actual time=15815.045..15815.047 rows=10 loops=1)</td></tr>
<tr><td>2</td><td>-> Sort (cost=3522534.11..3530581.53 rows=3218969 width=27) (actual time=15815.044..15815.045 rows=10 loops=1)</td></tr>
<tr><td>3</td><td>Sort Key: (count(t4."authorID")) DESC</td></tr>
<tr><td>4</td><td>Sort Method: top-N heapsort Memory: 25kB</td></tr>
<tr><td>5</td><td>-> GroupAggregate (cost=3388593.96..3452973.34 rows=3218969 width=27) (actual time=15730.679..15800.199 rows=100836 loops=1)</td></tr>
<tr><td>6</td><td>Group Key: t4."authorID", t2."authorName"</td></tr>
<tr><td>7</td><td>-> Sort (cost=3388593.96..3396641.38 rows=3218969 width=19) (actual time=15730.671..15760.202 rows=180754 loops=1)</td></tr>
<tr><td>8</td><td>Sort Key: t4."authorID", t2."authorName"</td></tr>
<tr><td>9</td><td>Sort Method: external merge Disk: 4936kB</td></tr>
<tr><td>10</td><td>-> Hash Join (cost=2703679.69..2908618.91 rows=3218969 width=19) (actual time=13237.256..15623.152 rows=180754 loops=1)</td></tr>
<tr><td>11</td><td>Hash Cond: (t4."authorID" = t2."authorID")</td></tr>
<tr><td>12</td><td>-> Merge Join (cost=2634501.57..2754286.26 rows=3218969 width=4) (actual time=12405.072..14027.717 rows=180754 loops=1)</td></tr>
<tr><td>13</td><td>Merge Cond: (t4."pubID" = publications."pubID")</td></tr>
</table>

Note: Screenshot only show a partial of the overall result. Result has a total of 34 rows shown below.

'Limit (cost=3522534.11..3522534.13 rows=10 width=27) (actual time=15815.045..15815.047 rows=10 loops=1)'

' -> Sort (cost=3522534.11..3530581.53 rows=3218969 width=27) (actual time=15815.044..15815.045 rows=10 loops=1)'

' Sort Key: (count(t4."authorID")) DESC'

' Sort Method: top-N heapsort Memory: 25kB'

' -> GroupAggregate (cost=3388593.96..3452973.34 rows=3218969 width=27) (actual time=15730.679..15800.199 rows=100836 loops=1)'

' Group Key: t4."authorID", t2."authorName"'

' -> Sort (cost=3388593.96..3396641.38 rows=3218969 width=19) (actual time=15730.671..15760.202 rows=180754 loops=1)'

' Sort Key: t4."authorID", t2."authorName"'

' Sort Method: external merge Disk: 4936kB'

' -> Hash Join (cost=2703679.69..2908618.91 rows=3218969 width=19) (actual time=13237.256..15623.152 rows=180754 loops=1)'

' Hash Cond: (t4."authorID" = t2."authorID")'

' -> Merge Join (cost=2634501.57..2754286.26 rows=3218969 width=4) (actual time=12405.072..14027.717 rows=180754 loops=1)'</td>
<td>The execution of this Query begins by running a Limit Operation.
In the Sub Branches, Hash Join Operations are performed between t and t.
Sequential Scan Operations are also performed on authored tables,publications tables using filter (title ~~ '%Data%' AND year >= 2013 AND year <= 2017 AND "pubKey" ~~ '%conf/%'),authors tables.
The method(s) that were used for the Sort Operations are top-N heapsort,external merge,external sort,publications.title.
Merge Join(s) was used with conditons t."pubID" = publications."pubID".
The planning time of this query plan is 17.123 milliseconds and actual execution time of this query is 15849.499 milliseconds.</td>
</tr>
</table>

```
'                    Merge Cond: (t4."pubID" = publications."pubID")'

'                    ->  Sort  (cost=2259871.75..2295071.25 rows=14079800
width=8) (actual time=9375.326..10294.646 rows=11969062 loops=1)'

'                          Sort Key: t4."pubID"'

'                          Sort Method: external sort  Disk: 247624kB'

'                          ->  Seq Scan on authored t4  (cost=0.00..203098.00
rows=14079800 width=8) (actual time=0.020..1613.766 rows=14079600
loops=1)'

'                    ->  Materialize  (cost=374629.83..375822.74 rows=36705
width=4) (actual time=3029.385..3066.501 rows=180917 loops=1)'

'                          ->  Unique  (cost=374629.83..375363.93 rows=36705
width=168) (actual time=3029.378..3056.448 rows=46397 loops=1)'

'                                ->  Sort  (cost=374629.83..374721.59 rows=36705
width=168) (actual time=3029.377..3044.265 rows=46397 loops=1)'

'                                      Sort Key: publications."pubID",
publications."pubKey", publications.title, publications.year, publications.month,
publications.ee, publications.url'

'                                      Sort Method: external merge  Disk: 9824kB'

'                                      ->  Append  (cost=0.00..368833.41 rows=36705
width=168) (actual time=0.160..2988.112 rows=46397 loops=1)'

'                                            ->  Seq Scan on publications
(cost=0.00..184233.18 rows=20021 width=193) (actual time=0.160..1531.759
rows=27975 loops=1)'

'                                                  Filter: ((title ~~ '%Data%'::text) AND
(year >= 2013) AND (year <= 2017) AND ("pubKey" ~~ '%conf/%'::text))'

'                                                  Rows Removed by Filter: 3795674'

'                                            ->  Seq Scan on publications publications_1
(cost=0.00..184233.18 rows=16684 width=193) (actual time=625.403..1453.126
rows=18422 loops=1)'

'                                                  Filter: ((title ~~ '%Data%'::text) AND
(year >= 2013) AND (year <= 2017) AND ("pubKey" ~~ '%journals/%'::text))'

'                                                  Rows Removed by Filter: 3805227'

'                    ->  Hash  (cost=32455.83..32455.83 rows=2000183 width=19)
(actual time=818.423..818.423 rows=2000183 loops=1)'

'                    Buckets: 65536  Batches: 32  Memory Usage: 3746kB'

'                    ->  Seq Scan on authors t2  (cost=0.00..32455.83
rows=2000183 width=19) (actual time=0.024..315.229 rows=2000183 loops=1)'

'Planning time: 0.914 ms'

'Execution time: 15865.856 ms'
```

8

2.3.11 Question 8

	QUERY PLAN text
1	Append (cost=1.43..206816.10 rows=376408 width=13) (actual time=0.027..1818.705 rows=479194 loops=1)
2	-> Merge Join (cost=1.43..3419.39 rows=6195 width=15) (actual time=0.026..23.718 rows=6431 loops=1)
3	Merge Cond: (proceedings."proceedingsID" = publications."pubID")
4	-> Index Scan using "proceedingsID" on proceedings (cost=0.29..1282.45 rows=34674 width=15) (actual time=0.007..5.394 rows=34674 loops=1)
5	-> Index Scan using publications_pkey on publications (cost=0.43..216619.86 rows=683580 width=8) (actual time=0.012..14.925 rows=6432 loops=1)
6	Filter: (month = 'June'::text)
7	Rows Removed by Filter: 28245
8	-> Merge Join (cost=4.69..196027.38 rows=364018 width=13) (actual time=0.034..1748.129 rows=472763 loops=1)
9	Merge Cond: (inproceedings."inproceedingsID" = publications_1."pubID")
10	-> Index Scan using inproceedings_pkey on inproceedings (cost=0.43..71074.68 rows=2037417 width=13) (actual time=0.010..486.066 rows=2037417 loops=1)
11	-> Index Scan using publications_pkey on publications publications_1 (cost=0.43..216619.86 rows=683580 width=8) (actual time=0.018..993.666 rows=472764 loops=1)
12	Filter: (month = 'June'::text)
13	Rows Removed by Filter: 1564671

Note: Screenshot only show a partial of the overall result. Result has a total of 26 rows shown below.

'Append (cost=1.43..206816.10 rows=376408 width=13) (actual time=0.027..1818.705 rows=479194 loops=1)'

' -> Merge Join (cost=1.43..3419.39 rows=6195 width=15) (actual time=0.026..23.718 rows=6431 loops=1)'

' Merge Cond: (proceedings."proceedingsID" = publications."pubID")'

' -> Index Scan using "proceedingsID" on proceedings (cost=0.29..1282.45 rows=34674 width=15) (actual time=0.007..5.394 rows=34674 loops=1)'

' -> Index Scan using publications_pkey on publications (cost=0.43..216619.86 rows=683580 width=8) (actual time=0.012..14.925 rows=6432 loops=1)'

' Filter: (month = 'June'::text)'

' Rows Removed by Filter: 28245'

' -> Merge Join (cost=4.69..196027.38 rows=364018 width=13) (actual time=0.034..1748.129 rows=472763 loops=1)'

' Merge Cond: (inproceedings."inproceedingsID" = publications_1."pubID")'

' -> Index Scan using inproceedings_pkey on inproceedings (cost=0.43..71074.68 rows=2037417 width=13) (actual time=0.010..486.066 rows=2037417 loops=1)'

' -> Index Scan using publications_pkey on publications publications_1 (cost=0.43..216619.86 rows=683580 width=8) (actual time=0.018..993.666 rows=472764 loops=1)'

' Filter: (month = 'June'::text)'

' Rows Removed by Filter: 1564671'

' -> Subquery Scan on sub_p_conf_names_abv_100_publications_in_june (cost=3481.34..3605.24 rows=6195 width=15) (actual time=30.933..30.933 rows=0 loops=1)'

' -> HashAggregate (cost=3481.34..3543.29 rows=6195 width=104) (actual time=30.931..30.931 rows=0 loops=1)'

The execution of this Query begins by running an **Append Operation**.
Index Scan(s) are used on **proceedings table** using **"proceedingsID" key, publications using publications_pkey key with month = 'June' filter, inproceedings** using **inproceedings_pkey key, publications** using **publications_pkey key with month = 'June' filter, proceedings** using **"proceedingsID" key, publications** using **publications_pkey key with month = 'June' filter**.
Merge Join(s) was used with **conditons proceedings."proceedingsID" = publications."pubID",inproceedings."inproceedingsID" = publications."pubID"**.
The **planning time** of this query plan is **34.993 milliseconds** and **actual execution time** of this query is **1818.190 milliseconds**.

<table>
<tr>
<td></td>
<td>' Group Key: publications_2.title, publications_2.year, proceedings_1."proceedingsID"'

' Filter: (count(publications_2."pubID") > '100'::bigint)'

' Rows Removed by Filter: 6431'

' -> Merge Join (cost=1.43..3419.39 rows=6195 width=100) (actual time=0.072..25.890 rows=6431 loops=1)'

' Merge Cond: (proceedings_1."proceedingsID" = publications_2."pubID")'

' -> Index Scan using "proceedingsID" on proceedings proceedings_1 (cost=0.29..1282.45 rows=34674 width=19) (actual time=0.029..7.339 rows=34674 loops=1)'

' -> Index Scan using publications_pkey on publications publications_2 (cost=0.43..216619.86 rows=683580 width=81) (actual time=0.038..14.690 rows=6432 loops=1)'

' Filter: (month = 'June'::text)'

' Rows Removed by Filter: 28245'

'Planning time: 0.930 ms'

'Execution time: 1831.490 ms'</td>
<td></td>
</tr>
<tr>
<td>9a</td>
<td>2.3.12 Question 9a

<table>
<tr><th></th><th>QUERY PLAN
text</th></tr>
<tr><td>1</td><td>GroupAggregate (cost=731534.53..736471.54 rows=80815 width=15) (actual time=13887.378..14737.144 rows=28 loops=1)</td></tr>
<tr><td>2</td><td>Group Key: authors."authorName"</td></tr>
<tr><td>3</td><td>Filter: (count(DISTINCT publications.year) = 31)</td></tr>
<tr><td>4</td><td>Rows Removed by Filter: 83747</td></tr>
<tr><td>5</td><td>-> Sort (cost=731534.53..732910.81 rows=550515 width=19) (actual time=13884.038..14524.954 rows=588378 loops=1)</td></tr>
<tr><td>6</td><td>Sort Key: authors."authorName"</td></tr>
<tr><td>7</td><td>Sort Method: external merge Disk: 16880kB</td></tr>
<tr><td>8</td><td>-> Hash Join (cost=264788.01..667750.76 rows=550515 width=19) (actual time=4112.935..11428.287 rows=588378 loops=1)</td></tr>
<tr><td>9</td><td>Hash Cond: (authored."pubID" = publications."pubID")</td></tr>
<tr><td>10</td><td>-> Hash Join (cost=38940.48..411000.50 rows=568877 width=19) (actual time=1977.284..7760.790 rows=598872 loops=1)</td></tr>
<tr><td>11</td><td>Hash Cond: (authored."authorID" = authors."authorID")</td></tr>
<tr><td>12</td><td>-> Seq Scan on authored (cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.011..1712.179 rows=14079600 loops=1)</td></tr>
<tr><td>13</td><td>-> Hash (cost=37456.29..37456.29 rows=80815 width=19) (actual time=1976.398..1976.398 rows=83775 loops=1)</td></tr>
</table>

Note: Screenshot only show a partial of the overall result. Result has a total of 24 rows shown below.

'GroupAggregate (cost=731534.53..736471.54 rows=80815 width=15) (actual time=13887.378..14737.144 rows=28 loops=1)'

' Group Key: authors."authorName"'

' Filter: (count(DISTINCT publications.year) = 31)'

' Rows Removed by Filter: 83747'

' -> Sort (cost=731534.53..732910.81 rows=550515 width=19) (actual time=13884.038..14524.954 rows=588378 loops=1)'

' Sort Key: authors."authorName"'</td>
<td>The execution of this Query begins by running an Aggregate Operation using Sorted Strategy on the result sets of its sub branches. In the Sub Branches, Hash Join Operations are performed between authored and publications,authored and authors. Sequential Scan Operations are also performed on authored tables,authors tables using filter ("authorName" ~~* 'H%'),publications tables using filter (year >= 1987 AND year <= 2017).
The method that was used for the Sort Operation is external merge.
The planning time of this query plan is 5.724 milliseconds and actual execution time of this query is 13332.944 milliseconds.</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
' Sort Method: external merge Disk: 16880kB'

' -> Hash Join (cost=264788.01..667750.76 rows=550515 width=19) (actual time=4112.935..11428.287 rows=588378 loops=1)'

' Hash Cond: (authored."pubID" = publications."pubID")'

' -> Hash Join (cost=38940.48..411000.50 rows=568877 width=19) (actual time=1977.284..7760.790 rows=598872 loops=1)'

' Hash Cond: (authored."authorID" = authors."authorID")'

' -> Seq Scan on authored (cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.011..1712.179 rows=14079600 loops=1)'

' -> Hash (cost=37456.29..37456.29 rows=80815 width=19) (actual time=1976.398..1976.398 rows=83775 loops=1)'

' Buckets: 65536 Batches: 2 Memory Usage: 2635kB'

' -> Seq Scan on authors (cost=0.00..37456.29 rows=80815 width=19) (actual time=0.019..1947.000 rows=83775 loops=1)'

' Filter: ("authorName" ~~* 'H%'::text)'

' Rows Removed by Filter: 1916408'

' -> Hash (cost=165103.14..165103.14 rows=3702512 width=8) (actual time=2134.703..2134.703 rows=3705215 loops=1)'

' Buckets: 131072 Batches: 64 Memory Usage: 3303kB'

' -> Seq Scan on publications (cost=0.00..165103.14 rows=3702512 width=8) (actual time=0.043..1367.387 rows=3705215 loops=1)'

' Filter: ((year >= 1987) AND (year <= 2017))'

' Rows Removed by Filter: 118434'

'Planning time: 0.927 ms'

'Execution time: 14748.661 ms'
</td>
<td></td>
</tr>
<tr>
<td>9b</td>
<td>
2.3.13 Question 9b

<table>
<tr><th></th><th>QUERY PLAN
text</th></tr>
<tr><td>1</td><td>Nested Loop (cost=22055.37..179391.46 rows=9651 width=23) (actual time=588.962..2728.518 rows=12 loops=1)</td></tr>
<tr><td>2</td><td>-> Hash Join (cost=22054.94..174860.76 rows=9651 width=8) (actual time=588.920..2728.409 rows=12 loops=1)</td></tr>
<tr><td>3</td><td>Hash Cond: (authored."pubID" = publications."pubID")</td></tr>
<tr><td>4</td><td>-> Seq Scan on authored (cost=0.00..132665.95 rows=5344895 width=8) (actual time=64.327..1427.325 rows=14079600 loops=1)</td></tr>
<tr><td>5</td><td>-> Hash (cost=21968.64..21968.64 rows=6904 width=4) (actual time=38.840..38.840 rows=12 loops=1)</td></tr>
<tr><td>6</td><td>Buckets: 8192 Batches: 1 Memory Usage: 65kB</td></tr>
<tr><td>7</td><td>-> Bitmap Heap Scan on publications (cost=129.94..21968.64 rows=6904 width=4) (actual time=38.692..38.809 rows=12 loops=1)</td></tr>
<tr><td>8</td><td>Recheck Cond: (year = '1936'::text)</td></tr>
<tr><td>9</td><td>Heap Blocks: exact=12</td></tr>
<tr><td>10</td><td>-> Bitmap Index Scan on publications_year (cost=0.00..128.21 rows=6904 width=0) (actual time=38.670..38.670 rows=12 loops=1)</td></tr>
<tr><td>11</td><td>Index Cond: (year = '1936'::text)</td></tr>
<tr><td>12</td><td>-> Index Scan using authors_pkey on authors (cost=0.43..0.46 rows=1 width=19) (actual time=0.008..0.008 rows=1 loops=12)</td></tr>
<tr><td>13</td><td>Index Cond: ("authorID" = authored."authorID")</td></tr>
<tr><td>14</td><td>Planning time: 1.052 ms</td></tr>
<tr><td>15</td><td>Execution time: 2728.653 ms</td></tr>
</table>
</td>
<td>
The execution of this Query begins by running a Nested Loop Operation.In the Sub Branches, Hash Join Operations are performed between authored and publications.
Sequential Scan Operations are also performed on authored tables.
The table that Bitmap Heap Scan Operation was performed on is publications.
There are 1 instance of Bitmap Index Scan performed during this Query using condition(s) year = '1936'.
Index Scan(s) are used on authors table using authors_pkey key with "authorID" = authored."authorID" condition.
</td>
</tr>
</table>

	Note: Screenshot above shows the full result of 15 rows.	The **planning time** of this query plan is **4.091 milliseconds** and actual execution time of this query is **2190.546 milliseconds .**
10	**2.3.14 Question 10** QUERY PLAN text 1 Group (cost=585771.18..585999.84 rows=5059 width=19) (actual time=37595.425..37665.126 rows=78103 loops=1) 2 Group Key: articles_authored."authorName", articles_authored."authorID" 3 -> Sort (cost=585771.18..585847.40 rows=30488 width=19) (actual time=37595.421..37644.973 rows=92518 loops=1) 4 Sort Key: articles_authored."authorName", articles_authored."authorID" 5 Sort Method: external merge Disk: 2776kB 6 -> Hash Join (cost=441029.94..583500.44 rows=30488 width=19) (actual time=5298.805..36599.153 rows=92518 loops=1) 7 Hash Cond: (authored."pubID" = books."pubID") 8 -> Nested Loop (cost=440602.66..581432.95 rows=356088 width=23) (actual time=5293.310..35874.760 rows=2516388 loops=1) 9 -> Hash Join (cost=440602.22..454686.73 rows=50586 width=23) (actual time=5292.967..6170.552 rows=78103 loops=1) 10 Hash Cond: (articles_authored."authorName" = authors."authorName") 11 -> Subquery Scan on articles_authored (cost=371424.10..372435.82 rows=50586 width=19) (actual time=4590.528..4623.993 rows=78103 loops=1) 12 -> HashAggregate (cost=371424.10..371929.96 rows=50586 width=19) (actual time=4590.527..4618.457 rows=78103 loops=1) 13 Group Key: authors_1."authorID" **Note: Screenshot only show a partial of the overall result. Result has a total of 37 rows shown below.** 'Group (cost=585771.18..585999.84 rows=5059 width=19) (actual time=37595.425..37665.126 rows=78103 loops=1)' ' Group Key: articles_authored."authorName", articles_authored."authorID"' ' -> Sort (cost=585771.18..585847.40 rows=30488 width=19) (actual time=37595.421..37644.973 rows=92518 loops=1)' ' Sort Key: articles_authored."authorName", articles_authored."authorID"' ' Sort Method: external merge Disk: 2776kB' ' -> Hash Join (cost=441029.94..583500.44 rows=30488 width=19) (actual time=5298.805..36599.153 rows=92518 loops=1)' ' Hash Cond: (authored."pubID" = books."pubID")' ' -> Nested Loop (cost=440602.66..581432.95 rows=356088 width=23) (actual time=5293.310..35874.760 rows=2516388 loops=1)' ' -> Hash Join (cost=440602.22..454686.73 rows=50586 width=23) (actual time=5292.967..6170.552 rows=78103 loops=1)' ' Hash Cond: (articles_authored."authorName" = authors."authorName")' ' -> Subquery Scan on articles_authored (cost=371424.10..372435.82 rows=50586 width=19) (actual time=4590.528..4623.993 rows=78103 loops=1)' ' -> HashAggregate (cost=371424.10..371929.96 rows=50586 width=19) (actual time=4590.527..4618.457 rows=78103 loops=1)' ' Group Key: authors_1."authorID"' ' -> Nested Loop (cost=91620.51..371297.64 rows=50586 width=19) (actual time=1429.514..4544.088 rows=92518 loops=1)'	The execution of this Query begins by running a **Group Operation**. In the Sub Branches, **Hash Join Operations** are performed between **authored and books,articlesauthored and authors,authored and publications. Sequential Scan Operations are also performed on authored tables,books tables,authors tables**. The method that was used for the **Sort Operation** is **external merge**. **Index Scan(s)** are used on authors **table** using **authors_pkey key with "authorID" = authored_1."authorID" condition**. **Index Only Scan(s)** are used on **publications table** using **publications_pkey key with "pubID" = books_1."pubID" condition, authored** using **authored_pkey key with "authorID" = authors."authorID" condition**. The **planning time** of this query plan is **16.348 milliseconds** and actual execution time of this query is **29717.656 milliseconds**.

	' -> Hash Join (cost=91620.08..348023.19 rows=50586 width=4) (actual time=1429.461..3728.259 rows=92518 loops=1)' ' Hash Cond: (authored_1."pubID" = publications."pubID")' ' -> Seq Scan on authored authored_1 (cost=0.00..203098.00 rows=14079800 width=8) (actual time=0.017..1838.956 rows=14079600 loops=1)' ' -> Hash (cost=91448.25..91448.25 rows=13746 width=8) (actual time=39.109..39.109 rows=13746 loops=1)' ' Buckets: 16384 Batches: 1 Memory Usage: 665kB' ' -> Nested Loop (cost=0.43..91448.25 rows=13746 width=8) (actual time=0.261..36.833 rows=13746 loops=1)' ' -> Seq Scan on books books_1 (cost=0.00..255.46 rows=13746 width=4) (actual time=0.009..1.170 rows=13746 loops=1)' ' -> Index Only Scan using publications_pkey on publications (cost=0.43..6.62 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=13746)' ' Index Cond: ("pubID" = books_1."pubID")' ' Heap Fetches: 13746' ' -> Index Scan using authors_pkey on authors authors_1 (cost=0.43..0.45 rows=1 width=19) (actual time=0.008..0.009 rows=1 loops=92518)' ' Index Cond: ("authorID" = authored_1."authorID")' ' -> Hash (cost=32455.83..32455.83 rows=2000183 width=19) (actual time=700.992..700.992 rows=2000183 loops=1)' ' Buckets: 65536 Batches: 32 Memory Usage: 3629kB' ' -> Seq Scan on authors (cost=0.00..32455.83 rows=2000183 width=19) (actual time=0.016..246.570 rows=2000183 loops=1)' ' -> Index Only Scan using authored_pkey on authored (cost=0.44..1.84 rows=67 width=8) (actual time=0.123..0.374 rows=32 loops=78103)' ' Index Cond: ("authorID" = authors."authorID")' ' Heap Fetches: 2516388' ' -> Hash (cost=255.46..255.46 rows=13746 width=4) (actual time=5.438..5.438 rows=13746 loops=1)' ' Buckets: 16384 Batches: 1 Memory Usage: 612kB'	

	' -> Seq Scan on books (cost=0.00..255.46 rows=13746 width=4) (actual time=0.039..3.429 rows=13746 loops=1)' 'Planning time: 1.357 ms' 'Execution time: 37682.268 ms'	