Group: Shrey Chaudhary (2803175)
Omi Patel (2835823)
Subject: CIS 611

Supervised by Prof. - SS Chung

Project Report

Building a Simple Query Optimizer with Performance Evaluation Experiment on Query Rewrite Optimization

Goals:

- Create a basic optimizer to analyze and suggest optimized versions of SQL queries..
- Conduct systematic tests to measure the impact of query rewrite on query performance
- Investigate and implement various optimization strategies within the query optimizer.
- Measure performance metrics, compare against baseline, document implementation, and draw conclusions for future improvements.

Programming language
JAVASCRIPT

Digital Platform:-

Software:- Visual Studio Code
Operating System:- MacOS(Big Sur(version 14.1.2))
Memory:- 16 GB
Processor: - Apple M2

Query Q1:-

```
SELECT T1.x1, SUM(T2.x2)
FROM T1,T2 WHERE T2.x2 = ( SELECT SUM(T3.x3)
FROM T3 WHERE T1.x1 = T3.x3) GROUP BY T1.x1;
```

Here above query retrieves the values of the column x1 from table T1 and the sum of column x2 from table T2 where the condition T2.x2 is equal to the result of a subquery. The subquery calculates the sum of column x3 from table T3 where the values of T1.x1 match those in T3.x3. The result is then grouped by the values of T1.x1. Essentially, it combines data from multiple tables, comparing and aggregating values based on specified conditions, and presents the result in a tabular format with one row for each distinct value of T1.x1, along with the corresponding sum of T2.x2.

Rebuilding it to RQ1:-

SELECT T1.x1, SUM(T2.x2) FROM T1,T2, (Select T1.Rowid, SUM(T3.x3) From T1, T3 Where T1.x1 = T3.x3 Group By T1.Rowid) as Temp1 (rowid, x3) WHERE T2.x2 = Temp.x3 and T1.Rowid = Temp.rowid GROUP BY T1.x1;

Above query selects the values of T1.x1 and the sum of T2.x2 from tables T1 and T2, joining with a temporary table (Temp1) that calculates the sum of T3.x3 grouped by T1.Rowid. The result is filtered by conditions linking T2.x2 to Temp1.x3 and T1.Rowid to Temp1.rowid, then grouped by T1.x1.

A possible Query Execution Steps of Q1:

1. Join t1 t2
2. Join temp1 t3
3. Project temp2
4. GroupBy temp3

A possible Query Execution Steps of RQ1:

1. Join t1 t3
2. GroupBy temp0
3. Join t1 Temp1
4. Join t2 Temp2
5. Project temp3
6. GroupBy temp3

Given Join Methods:

- Tuple Nested Loop Join: TNL
- Page Nested Loop Join: PNL
- Block Nested Loop Join: with Buffer memory B =50: BNJM
- Sort Merge Join with buffer memory B=50: SMJM
- Hash Join with Buffer memory requirement B=50 pages for hash table: HJM
- Hash Join with less Buffer memory B= 30 pages : HJL
- Block Nested Loop Join: with less Buffer memory B =30: BNJL
- Sort Merge Join with less buffer memory B=30: SMJL

We have to calculate join cost for each join methods

Cost of TNL = LPages + (#Tuples_Per_Page_In_L * LPages) * RPages

Cost of PNL = LPages + (LPages * RPages)

Cost of BNL = LPages + (LPages / BlockSize) * RPages

SMJL Join cost = 2 (M + N) (MlogM)+ M + N

If we have Buffer size B^2 > Size of bigger Table SMJM or SMJL Join cost = 3 * (M + N)

HJM cost: 3(M + N)

Given input:-

bufferMemoryWithBuff = 50;
bufferMemoryWithlessBuff = 30;
Page Size=4096;
Block Size=100;
Table Size:-
t1s = 20;
t1p = 1000;
t2s = 40;
t2p = 500;
t3s = 100;
t3p = 2000;
Q1t = "Join t1 t2\r\n" +"Join temp1 t3\r\n" +"Project temp2\r\n" +"GroupBy temp3";
RQ1t = "Join t1 t3\r\n" +"GroupBy temp0\r\n" +"Join t1 Temp1\r\n" +"Join t2 Temp2\r\n" +"Project temp3\r\n" + "GroupBy temp3";

Explanation of Source Code :

Classes:

1. ShOm_COSTData:

Purpose:

Represents a data structure for storing join-related information.

Properties:

ShOm_joinCost: The cost of the join operation.
ShOm_isReverseOrder: Indicates whether the join order is reversed.
ShOm_joinType: Stores the type of join in the code.

2**. ShOm_QueryProcessor:**

Purpose:

Static class containing properties and methods for query processing.

Properties:

Constants defining buffer memory sizes, page sizes, and block sizes.

Static variables to configure the processing behavior.

Query strings for two different query plans (ShOm_Q1t and RShOm_Q1t).

Methods:

main: Entry point for the query processor.

ShOm_processTheQuery: Orchestrates the processing of query plans, comparing their costs.

displayShOm_output: Displays the output, indicating the best query plan.

Methods for processing different types of joins (TNL, PNL, BNL, SMJ, Hash).

Utility methods for calculating group-by costs and formatting query processing time.

Execution :

The main function initiates the query processing by calling ShOm_processTheQuery with specified parameters.

The main function creates the entry point and calls the ShOm_processTheQuery function with the appropriate parameters to start query processing. This start probably makes it easier for the ShOm_processTheQuery function to carry out its query processing logic. The way ShOm_processTheQuery is implemented determines each aspect of the query processing and its results.

Query Processing :

ShOm_processQ1:

This function is dedicated to handling the Q1 query plan and involves a comprehensive process of evaluating and comparing costs associated with each query execution step.

It meticulously calculates costs for various operations within the plan, such as joins, ordering, and other relevant steps.

The output of this function provides a detailed breakdown of each step in the Q1 query plan, including crucial information like join types, ordering specifications, and the corresponding costs associated with each operation. Additionally, it furnishes a summary that encompasses the total disk I/O cost and the overall query processing cost for the Q1 query.

ShOm_processRQ1:

This function mirrors the ShOm_processQ1 but is specifically tailored for the RQ1 query plan, demonstrating a modular and adaptable design to handle distinct queries.

Similar to ShOm_processQ1, it systematically processes the RQ1 query plan, evaluating costs for individual steps, and providing detailed insights into the nature of each operation.

The output of ShOm_processRQ1 encompasses a breakdown of RQ1 query plan steps, specifying join types, ordering details, and associated costs. Additionally, it offers a comprehensive summary that includes the total disk I/O cost and the overall query processing cost for the RQ1 query.

Join Algorithms:

Implements several join algorithms, including Tuple Nested Loop Join, Page Nested Loop Join, Block Nested Loop Join, Sort Merge Join, and Hash Join.

Each algorithm has its own cost calculation logic and is chosen based on specific conditions.

Output:

The query processing system's produced output processes is essential for revealing the costs and procedures involved in carrying out each query plan step. The system carefully documents and displays comprehensive information about the actions taken at each stage as it moves through the query plan. This data frequently contains specifics like the kind of joins used, the ordering standards, and the associated expenses.

The comprehensive result provides insight into the system's decision-making process and is an invaluable tool for comprehending the nuances of query execution. With this output, analysts and developers can evaluate the effectiveness of selected algorithms, spot possible bottlenecks, and improve query plan performance.

Additionally, the output at the end works as a summary that covers up all of the expenses incurred during the execution of the query plan. The query plan that is deemed most optimal in terms of total cost is determined in large part by this summary. The assessment of overall expenses facilitates the decision-making process regarding the optimal execution approach,

thereby supporting the continuous endeavors to improve query performance and overall system efficiency. Essentially, the final cost-based decision and the comprehensive output give developers and administrators the necessary information to adjust and optimize the query processing system for better performance.

Observations:

1. The code seems to be a simplified educational representation of a query processor.
2. Certain calculations and comparisons may require further review for correctness and efficiency.
3. The static nature of the class may limit flexibility for handling multiple instances or scenarios concurrently.

Source code and screenshots of source code and output are shown below:

Source Code:

```
class ShOm_COSTData {
   constructor() {
      this.ShOm_joinCost = 0;
      this.ShOm_isReverseOrder = false;
      this.ShOm_joinType = '';
   }
}

class ShOm_QueryProcessor {
   static ShOm_ShOm_bufferMemoryWithBuff = 50;
   static ShOm_ShOm_bufferMemoryWithlessBuff = 30;
   static ShOm_t1TupPerPage;
   static ShOm_t2TupPerPage;
   static ShOm_t3TupPerPage;
   static ShOm_pageSize = 4096;
   static ShOm_blockSize = 100;
   static ShOm_output = '';
   static ShOm_isSortergeDone = false;
   static ShOm_t1s = 20;
   static ShOm_t1p = 1000;
   static ShOm_t2s = 40;
   static ShOm_t2p = 500;
   static ShOm_t3s = 100;
   static ShOm_t3p = 2000;
   static ShOm_Q1t = "Join t1 t2\r\n" + "Join temp1 t3\r\n" + "Project temp2\r\n" + "GroupBy
temp3";
   static RShOm_Q1t = "Join t1 t3\r\n" + "GroupBy temp0\r\n" + "Join t1 Temp1\r\n" + "Join t2
Temp2\r\n"
```

```
      + "Project temp3\r\n" + "GroupBy temp3";

   static main() {
      ShOm_QueryProcessor.ShOm_processTheQuery(ShOm_QueryProcessor.ShOm_t1s,
ShOm_QueryProcessor.ShOm_t1p, ShOm_QueryProcessor.ShOm_t2s,
ShOm_QueryProcessor.ShOm_t2p, ShOm_QueryProcessor.ShOm_t3s,
ShOm_QueryProcessor.ShOm_t3p);
   }

   static ShOm_processTheQuery(ShOm_t1s, ShOm_t1pages, ShOm_t2s, ShOm_t2pages,
ShOm_t3s, ShOm_t3pages) {
      ShOm_QueryProcessor.ShOm_t1TupPerPage = ShOm_QueryProcessor.ShOm_pageSize /
ShOm_t1s;
      ShOm_QueryProcessor.ShOm_t2TupPerPage = ShOm_QueryProcessor.ShOm_pageSize /
ShOm_t2s;
      ShOm_QueryProcessor.ShOm_t3TupPerPage = ShOm_QueryProcessor.ShOm_pageSize /
ShOm_t3s;
      console.log("Go with Q1 query");
      let cost1 = ShOm_QueryProcessor.ShOm_processQ1(ShOm_t1pages, ShOm_t2pages,
ShOm_t3pages);
      console.log("Go with RQ1 query");
      let cost2 = ShOm_QueryProcessor.ShOm_processRQ1(ShOm_t1pages, ShOm_t2pages,
ShOm_t3pages);
      ShOm_QueryProcessor.displayShOm_output(cost1, cost2);
   }

   static displayShOm_output(cost1, cost2) {
      ShOm_QueryProcessor.ShOm_output += "\n---Best Query Plan:---\n";
      try {
         if (cost1 > cost2) {
            ShOm_QueryProcessor.ShOm_output += "\nThe Best Query Plan is RQ1 because cost
of RQ1 is less than cost of Q1\n";
            console.log(ShOm_QueryProcessor.ShOm_output);
         } else {
            ShOm_QueryProcessor.ShOm_output += "\nThe Best Query Plan is Q1 since cost of
RQ1 is more than cost of Q1\n";
            console.log(ShOm_QueryProcessor.ShOm_output);
         }
      } catch (ex) {
      }
   }

   static ShOm_findMinimumJoinCost(leftTable, rightTable, obj, type) {
      try {
         let cost = 0;
```

```
        if (type === 1)
            cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable,
ShOm_QueryProcessor.ShOm_t1TupPerPage, obj);
        else if (type === 2)
            cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable,
ShOm_QueryProcessor.ShOm_t3TupPerPage, obj);
        else if (type === 3)
            cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable,
ShOm_QueryProcessor.ShOm_t1TupPerPage, obj);
        else if (type === 4)
            cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable,
ShOm_QueryProcessor.ShOm_t1TupPerPage, obj);
        else if (type === 5)
            cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable,
ShOm_QueryProcessor.ShOm_t2TupPerPage, obj);
        cost = ShOm_QueryProcessor.ShOm_PNLJoinCost(leftTable, rightTable, obj);

        cost = ShOm_QueryProcessor.ShOm_hashJoinCostWithBuffer(leftTable, rightTable,
obj);
        cost = ShOm_QueryProcessor.ShOm_hashJoinCostWithLessBuffer(leftTable,
rightTable, obj);
        cost = ShOm_QueryProcessor.SShOm_MJJoinCostWithBuf(leftTable, rightTable, obj);
        cost = ShOm_QueryProcessor.ShOm_SMJJoinCostWithLessBuf(leftTable, rightTable,
obj);
        cost = ShOm_QueryProcessor.ShOm_BNLJoinCostWithBuf(leftTable, rightTable, obj);
        cost = ShOm_QueryProcessor.ShOm_BNLJoinCostWithLessBuf(leftTable, rightTable,
obj);
    } catch (ex) {
    }
}

static ShOm_GetGroupByCost(table1, table2, isSMJDone) {
    try {
        if (isSMJDone) {
            return 2 * (table1 + table2);
        } else {
            return 3 * (table1 + table2);
        }
    } catch (ex) {
    }
    return 0;
}

static ShOm_joinTables(table1, table2) {
    return table1 * table2;
}
```

```
   static ShOm_QueryProcessoringCost(diskIOCost) {
      try {
         let ioCost = BigInt(diskIOCost) * BigInt(12) / BigInt(1000);
         let totalSeconds = parseInt(ioCost);
         let seconds = totalSeconds % 60;
         let totalMinutes = totalSeconds / 60;
         let minutes = totalMinutes % 60;
         let hours = totalMinutes / 60;
         return hours + " HR(S) " + minutes + " MIN(S) " + seconds + " SEC(S)";
      } catch (ex) {
      }
      return "";
   }

   static ShOm_processQ1(ShOm_t1pages, ShOm_t2pages, ShOm_t3pages) {
      try {
         ShOm_QueryProcessor.ShOm_output += "\n---Cost of Q1:---\n";
         let totalCost = 0;
         let tempTable = 0;
         for (let line of ShOm_QueryProcessor.ShOm_Q1t.split("\r\n")) {
            if (line.toLowerCase().includes("join")) {
               let elements = line.split(" ");
               let leftTable = elements[1];
               let rightTable = elements[2];
               if (leftTable === "t1" && rightTable === "t2") {
                  if (ShOm_t1pages > ShOm_t2pages) {
                     let select = new ShOm_COSTData();
                     ShOm_QueryProcessor.ShOm_findMinimumJoinCost(ShOm_t2pages,
ShOm_t1pages, select, 1);
                     tempTable = ShOm_QueryProcessor.ShOm_joinTables(ShOm_t1pages,
ShOm_t2pages);
                     tempTable = (tempTable * 15) / 100;
                     console.log("The temp value is " + tempTable);
                     totalCost += select.joinCost;
                     ShOm_QueryProcessor.ShOm_output += elements[0];
                     if (select.isReverseOrder) {
                        ShOm_QueryProcessor.ShOm_output += " " + elements[2] + " " +
elements[1];
                     } else {
                        ShOm_QueryProcessor.ShOm_output += " " + elements[1] + " " +
elements[2];
                     }
                     ShOm_QueryProcessor.ShOm_output += "-->Cost: " + select.joinCost + ", Join
type:  " + select.joinType + "\n";
                  }
```

```
            } else if (leftTable === "temp1") {
                                let select2 = new ShOm_COSTData();
                ShOm_QueryProcessor.ShOm_TNLJoinCost(ShOm_t3pages, tempTable,
ShOm_QueryProcessor.ShOm_t3TupPerPage, select2);
                tempTable = (tempTable * 10 * 20) / 10000;
                totalCost += select2.joinCost;
                ShOm_QueryProcessor.ShOm_output += elements[0];
                if (select2.isReverseOrder) {
                   ShOm_QueryProcessor.ShOm_output += " " + elements[2] + " " + elements[1];
                } else {
                   ShOm_QueryProcessor.ShOm_output += " " + elements[1] + " " + elements[2];
                }
                ShOm_QueryProcessor.ShOm_output += "-->Cost: " + select2.joinCost + ", Join
type: " + select2.joinType + "\n";
             }
          }
          if (line.toLowerCase().includes("project")) {
             let elements = line.split(" ");
             ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "--
>Cost is too small, we will neglect it.\n";
          }
          if (line.toLowerCase().includes("groupby")) {
             let elements = line.split(" ");
             let groupByCost = ShOm_QueryProcessor.ShOm_GetGroupByCost(tempTable,
ShOm_t3pages, false);
             totalCost += groupByCost;
             ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "--
>Cost: " + groupByCost + "\n";
          }
       }
       ShOm_QueryProcessor.ShOm_output += "Total Disk I/O Cost is :- " + totalCost + "\n";
       ShOm_QueryProcessor.ShOm_output += "Query Processing Cost :- " +
ShOm_QueryProcessor.ShOm_QueryProcessoringCost(totalCost) + "\n";
       return totalCost;
    } catch (ex) {
    }
    return 0;
 }

 static ShOm_processRQ1(ShOm_t1pages, ShOm_t2pages, ShOm_t3pages) {
    try {
       ShOm_QueryProcessor.ShOm_output += "\n---Cost of RQ1--- :- \n";
       let totalShOm_QueryProcessoringCost = 0;
       let tempTable = 0;
       for (let line of ShOm_QueryProcessor.RShOm_Q1t.split("\r\n")) {
          if (line.toLowerCase().includes("join")) {
```

```
                let elements = line.split(" ");
                let leftTable = elements[1];
                let rightTable = elements[2];
                if (leftTable === "t1" && rightTable === "t3") {
                    if (ShOm_t3pages > ShOm_t1pages) {
                        let select = new ShOm_COSTData();
                        ShOm_QueryProcessor.ShOm_findMinimumJoinCost(ShOm_t3pages,
ShOm_t1pages, select, 3);
                        tempTable = ShOm_QueryProcessor.ShOm_joinTables(ShOm_t3pages,
ShOm_t1pages);
                        tempTable = (tempTable * 20) / 100;
                        totalShOm_QueryProcessoringCost += select.joinCost;
                        if (select.isReverseOrder) {
                            ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[2] +
" " + elements[1];
                        } else {
                            ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] +
" " + elements[2];
                        }
                        ShOm_QueryProcessor.ShOm_output += "-->Cost: " + select.joinCost + ", Join
type: " + select.joinType + "\n";
                    }
                } else if (leftTable === "t1") {
                    let select1 = new ShOm_COSTData();
                    ShOm_QueryProcessor.ShOm_findMinimumJoinCost(ShOm_t1pages,
tempTable, select1, 4);
                    tempTable = (tempTable * 15) / 100;
                    totalShOm_QueryProcessoringCost += select1.joinCost;
                    if (select1.isReverseOrder) {
                        ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[2] + " "
+ elements[1];
                    } else {
                        ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + " "
+ elements[2];
                    }
                    ShOm_QueryProcessor.ShOm_output += "-->Cost: " + select1.joinCost + ", Join
type: " + select1.joinType + "\n";
                } else if (leftTable === "t2") {
                    let select2 = new ShOm_COSTData();
                    ShOm_QueryProcessor.ShOm_findMinimumJoinCost(ShOm_t2pages,
tempTable, select2, 5);
                    tempTable = (tempTable * 10) / 100;
                    totalShOm_QueryProcessoringCost += select2.joinCost;
                    if (select2.isReverseOrder) {
                        ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[2] + " "
+ elements[1] + "-->Cost: "
```

```
                        + select2.joinCost + ", Join type: " + select2.joinType + "" + "\n";
                } else {
                    ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + " "
+ elements[2] + "-->Cost: "
                        + select2.joinCost + ", Join type: " + select2.joinType + "\n";
                }
            }
        }
        if (line.toLowerCase().includes("project")) {
            let elements = line.split(" ");
            ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "--
>Cost is too small, neglect it\n";
        }
        if (line.toLowerCase().includes("groupby")) {
            let elements = line.split(" ");
            let groupByCost = 0;
            if (elements[1] === "temp0") {
                groupByCost = ShOm_QueryProcessor.ShOm_GetGroupByCost(ShOm_t1pages,
ShOm_t3pages, ShOm_QueryProcessor.ShOm_isSortergeDone);
            } else {
                groupByCost = ShOm_QueryProcessor.ShOm_GetGroupByCost(tempTable,
ShOm_t3pages, ShOm_QueryProcessor.ShOm_isSortergeDone);
            }
            totalShOm_QueryProcessoringCost += groupByCost;
            ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "--
>Cost: " + groupByCost + "\n";
        }
    }
    ShOm_QueryProcessor.ShOm_output += "Total Disk I/O Cost is :- " +
totalShOm_QueryProcessoringCost + "\n";
    ShOm_QueryProcessor.ShOm_output += "Query Processing Cost :- " +
ShOm_QueryProcessor.ShOm_QueryProcessoringCost(totalShOm_QueryProcessoringCost) +
"\n";
    return totalShOm_QueryProcessoringCost;
  } catch (ex) {
  }
  return 0;
}

static ShOm_TNLJoinCost(leftTable, rightTable, tuplesPerPage, obj) {
  try {
    let cost = (leftTable + tuplesPerPage * leftTable * rightTable);

    if ((cost >= 0)) {
      obj.joinCost = cost;
      obj.joinType = "TNL";
```

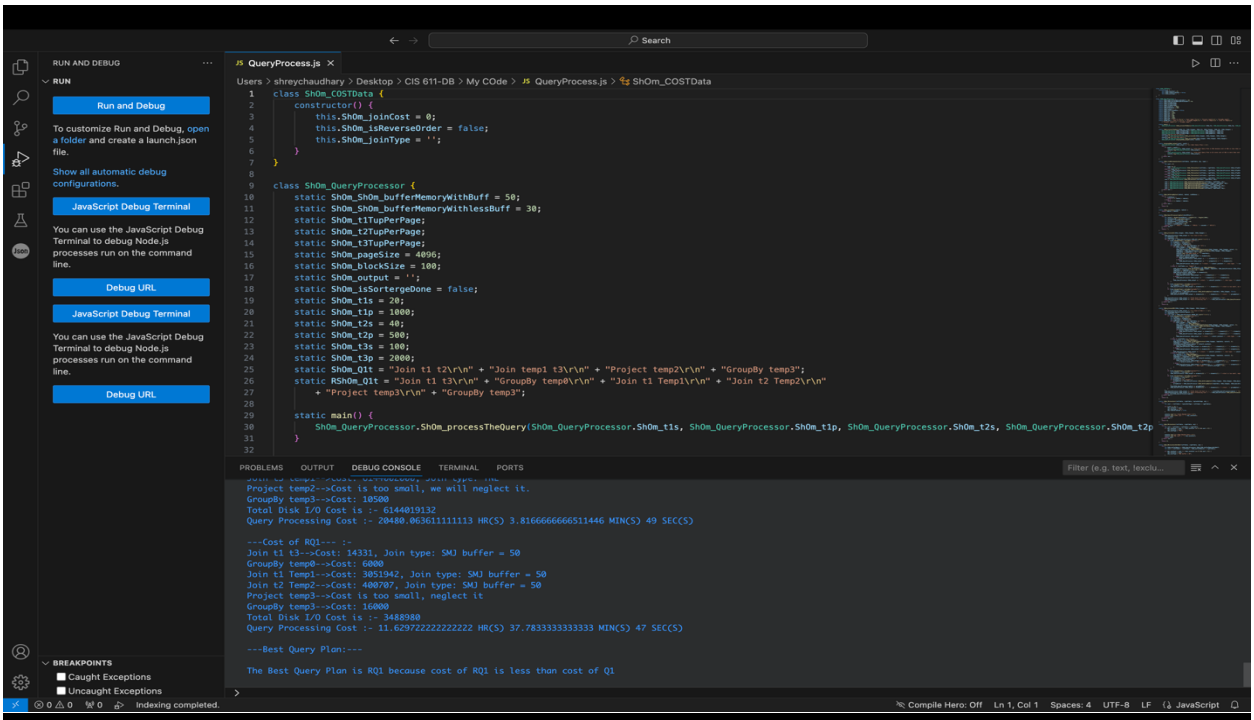
```
            obj.isReverseOrder = true;
        }

        console.log("\n---Tuple Nested Join---\n");
        console.log("\nTNL cost :- " + obj.joinCost);
        return cost;
    } catch (ex) {
    }
    return 0;
}

static ShOm_PNLJoinCost(leftTable, rightTable, obj) {
    try {
        let cost = leftTable + leftTable * rightTable;
        if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
            obj.joinCost = cost;
            obj.joinType = "PNL";
        }

        console.log("\n---Page Nested Join---\n");
        console.log("PNL cost :- " + obj.joinCost);
        return cost;
    } catch (ex) {
    }
    return 0;
}

static ShOm_BNLJoinCostWithBuf(leftTable, rightTable, obj) {
    try {
        let ShOm_bufferMemory =
ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithBuff;
        let cost = (leftTable + (leftTable / ShOm_bufferMemory) * rightTable);

        if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
            obj.joinCost = cost;
            obj.joinType = "BNL buffer = 50";
        }
        console.log("\n Block Nested Loop Join with buffer = 50 \n");
        console.log("BNL cost: " + obj.joinCost);
        return cost;
    } catch (ex) {
    }
    return 0;
}

static ShOm_BNLJoinCostWithLessBuf(leftTable, rightTable, obj) {
```

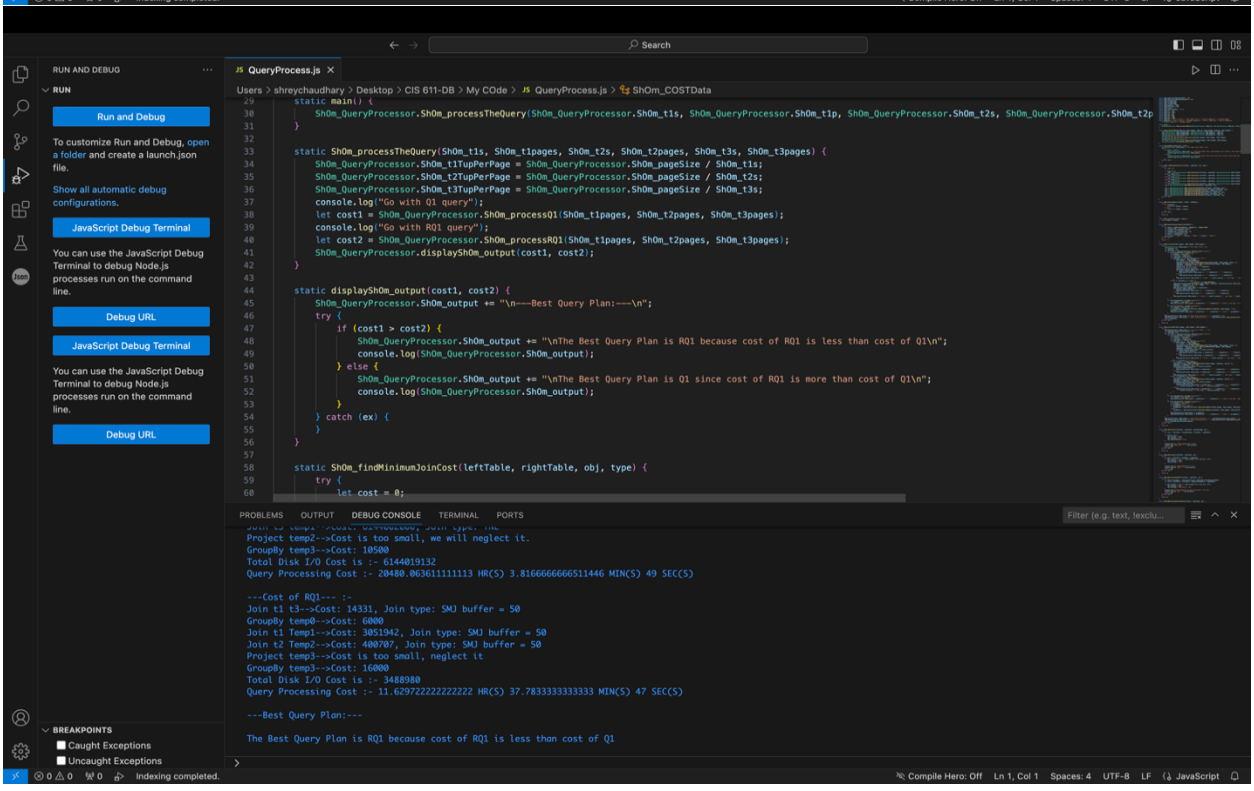
```
    try {
        let ShOm_bufferMemory =
ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithlessBuff;
        let cost = (leftTable + (leftTable / ShOm_bufferMemory) * rightTable);
        if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
            obj.joinCost = cost;
            obj.joinType = "BNL buffer = 30";
            ShOm_QueryProcessor.ShOm_isSortergeDone = true;
            obj.isReverseOrder = false;
        }
        console.log("\nBlock Nested Loop Join with buffer = 30\n");
        console.log("BNL cost: " + obj.joinCost);
        return cost;
    } catch (ex) {
    }
    return 0;
  }

  static SShOm_MJJoinCostWithBuf(leftTable, rightTable, obj) {
    try {
        let cost;
        let ShOm_bufferMemory =
ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithBuff;
        let ShOm_mMultiplier = (Math.log10(leftTable / ShOm_bufferMemory)) /
(Math.log10(ShOm_bufferMemory - 1));
        let ShOm_nMultiplier = (Math.log10(rightTable / ShOm_bufferMemory)) /
(Math.log10(ShOm_bufferMemory - 1));
        cost = Math.ceil(
            2 * leftTable * (1 + ShOm_mMultiplier) + 2 * rightTable * (1 + ShOm_nMultiplier) +
leftTable + rightTable);
        if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
            obj.joinCost = cost;
            obj.joinType = "SMJ buffer = 50";
            obj.isReverseOrder = false;
            ShOm_QueryProcessor.ShOm_isSortergeDone = true;
        }
        console.log("\n Sort Merge Join with buffer = 50\n");
        console.log("SMJ cost: " + obj.joinCost);
        return cost;
    } catch (ex) {
    }
    return 0;
  }

  static ShOm_SMJJoinCostWithLessBuf(leftTable, rightTable, obj) {
    try {
```

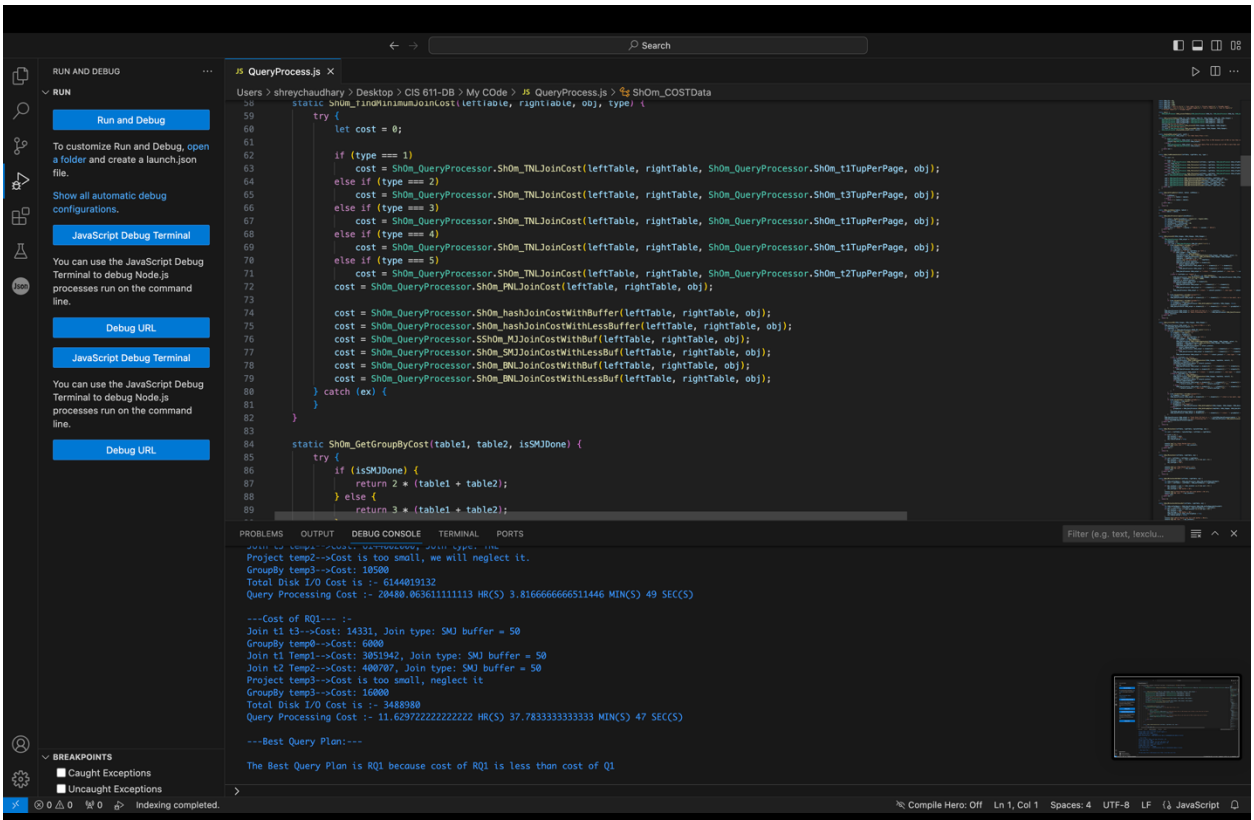
```
        let cost;
        let ShOm_bufferMemory =
ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithlessBuff;
        let ShOm_mMultiplier = Math.log10(leftTable / ShOm_bufferMemory) /
Math.log10(ShOm_bufferMemory - 1);
        let ShOm_nMultiplier = Math.log10(rightTable / ShOm_bufferMemory) /
Math.log10(ShOm_bufferMemory - 1);
        cost = Math.ceil(
           (2 * leftTable * (1 + ShOm_mMultiplier) + 2 * rightTable * (1 + ShOm_nMultiplier) +
leftTable + rightTable));
        if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
           obj.joinCost = cost;
           obj.joinType = "SMJ buffer = " + ShOm_bufferMemory;
           obj.isReverseOrder = false;
           ShOm_QueryProcessor.ShOm_isSortergeDone = true;
        }
        console.log("\nSort Merge Join with less buffer = 30\n");
        console.log("SMJ cost: " + obj.joinCost);
        return cost;
     } catch (ex) {
     }
     return 0;
  }

  static ShOm_hashJoinCostWithBuffer(leftTable, rightTable, obj) {
     try {
        let ShOm_bufferMemory =
ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithBuff;
        let cost;
        let multiplier = Math.log10((leftTable + rightTable) / (ShOm_bufferMemory - 1))
           / (Math.log10(ShOm_bufferMemory - 1));
        cost = Math.ceil(2 * (leftTable + rightTable) * (1 + multiplier) + leftTable + rightTable);
        if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
           obj.joinCost = cost;
           obj.joinType = "Hash Join buffer = " + ShOm_bufferMemory;
        }
        console.log("\n Hash Join with buffer = 50\n");
        console.log("HashJoin cost: " + obj.joinCost);
        return cost;
     } catch (ex) {
     }
     return 0;
  }

  static ShOm_hashJoinCostWithLessBuffer(leftTable, rightTable, obj) {
     try {
```

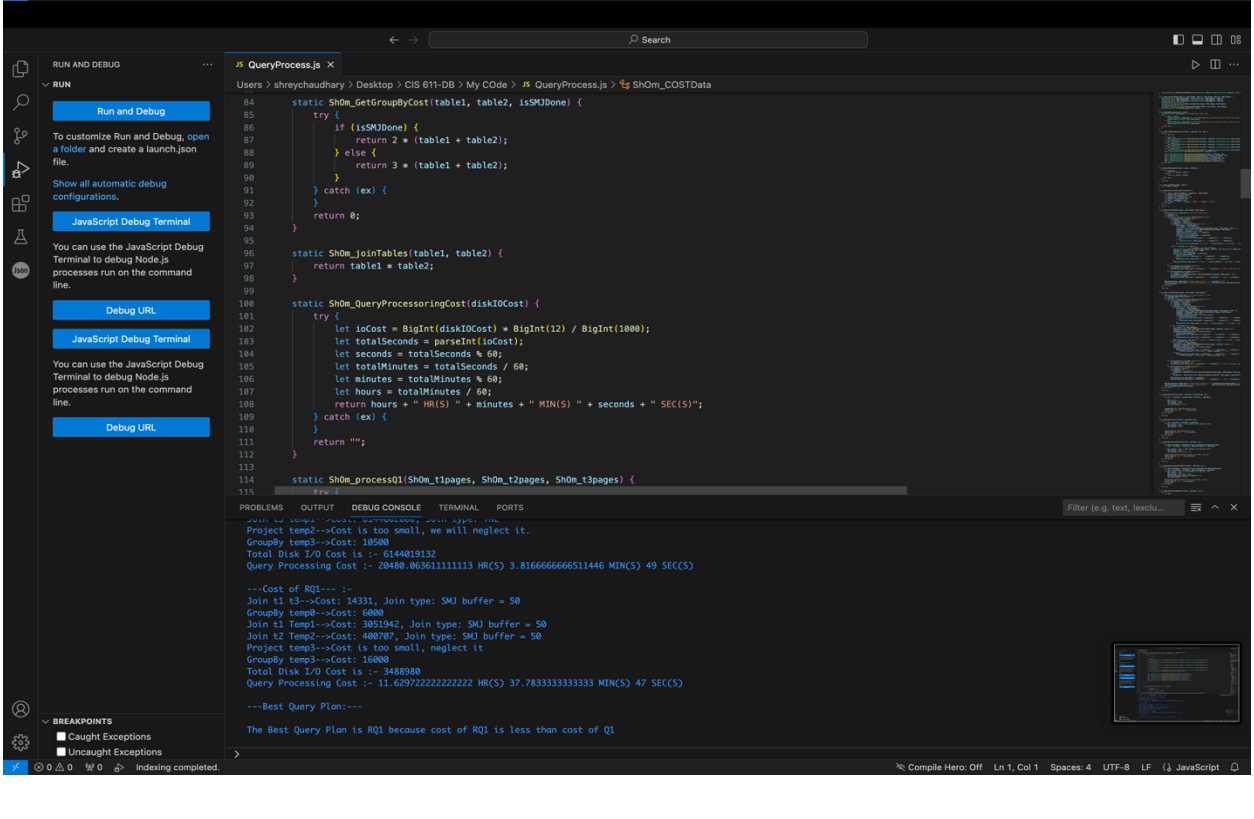
```
        let ShOm_bufferMemory =
ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithlessBuff;
        let multiplier = Math.log10((leftTable + rightTable) / (ShOm_bufferMemory - 1))
           / (Math.log10(ShOm_bufferMemory - 1));
        let cost;
        cost = Math.ceil(2 * (leftTable + rightTable) * (1 + multiplier) + leftTable + rightTable);
        if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
           obj.joinCost = cost;
           obj.joinType = "Hash Join buffer = " + ShOm_bufferMemory;
        }
        console.log("\nHash Join with less buffer = 30\n");
        console.log("HashJoin cost: " + obj.joinCost);
        return cost;
      } catch (ex) {
      }
      return 0;
   }
}


ShOm_QueryProcessor.main();
```

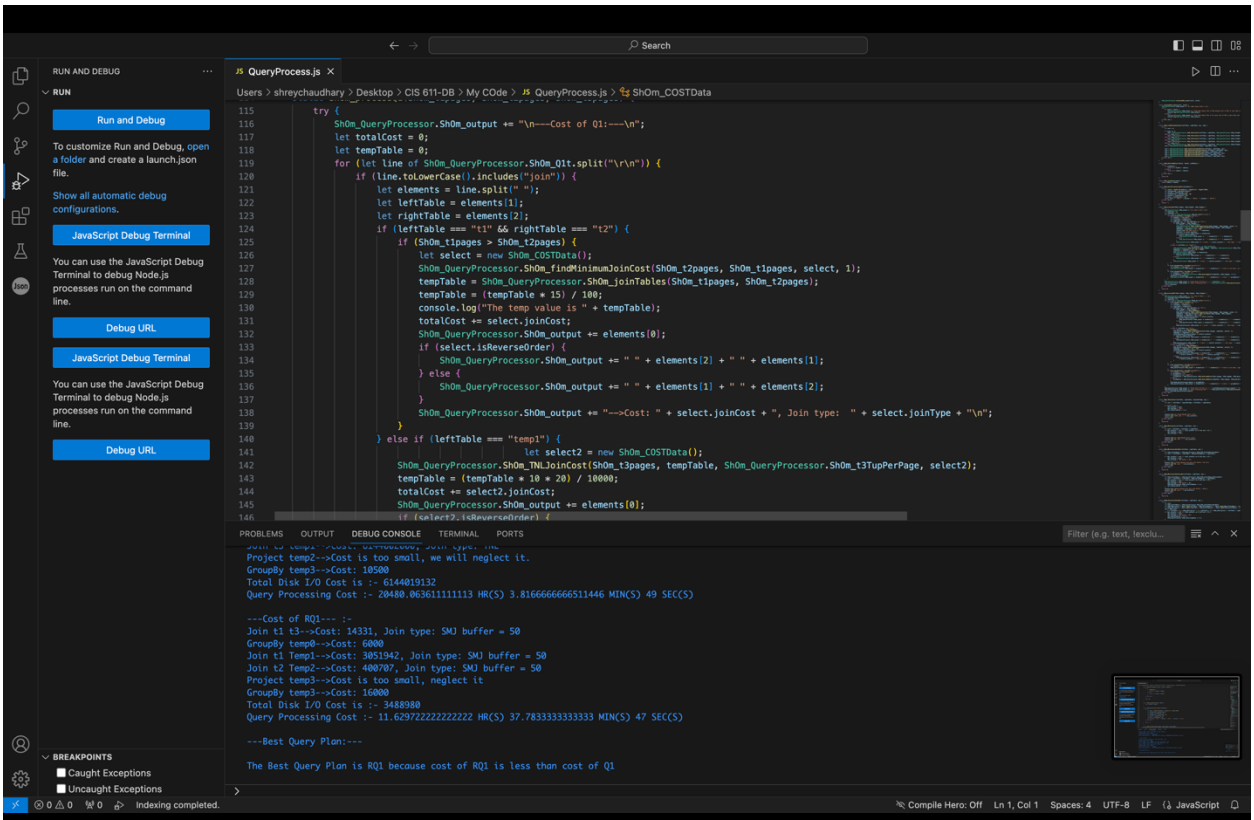
Screenshots of code :

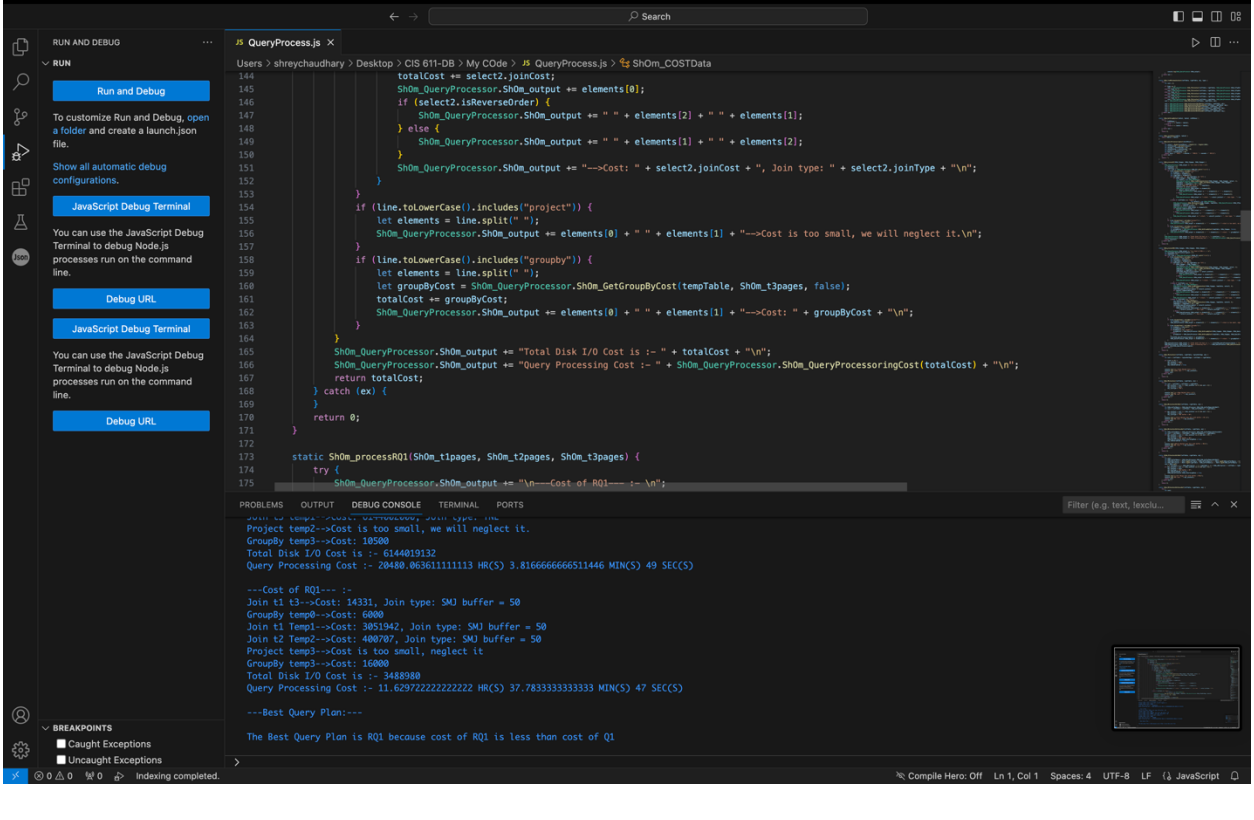


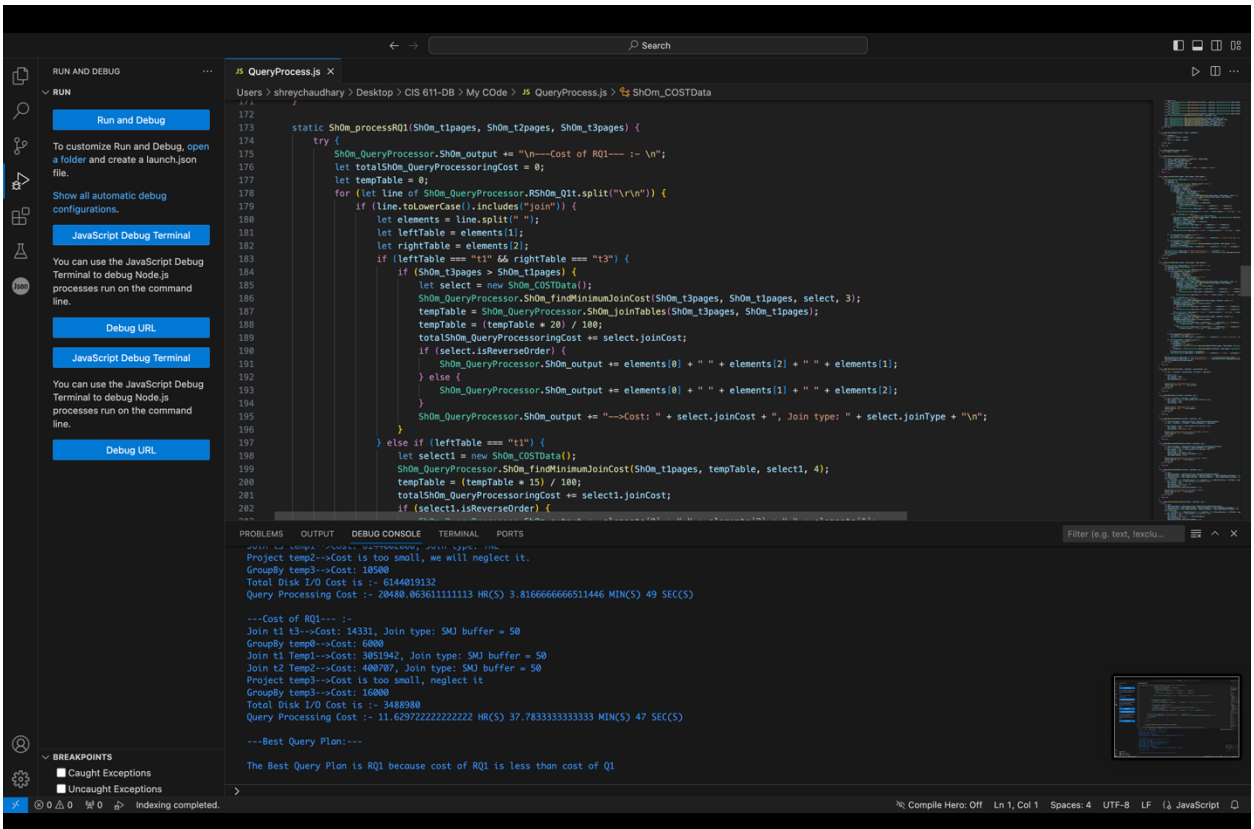


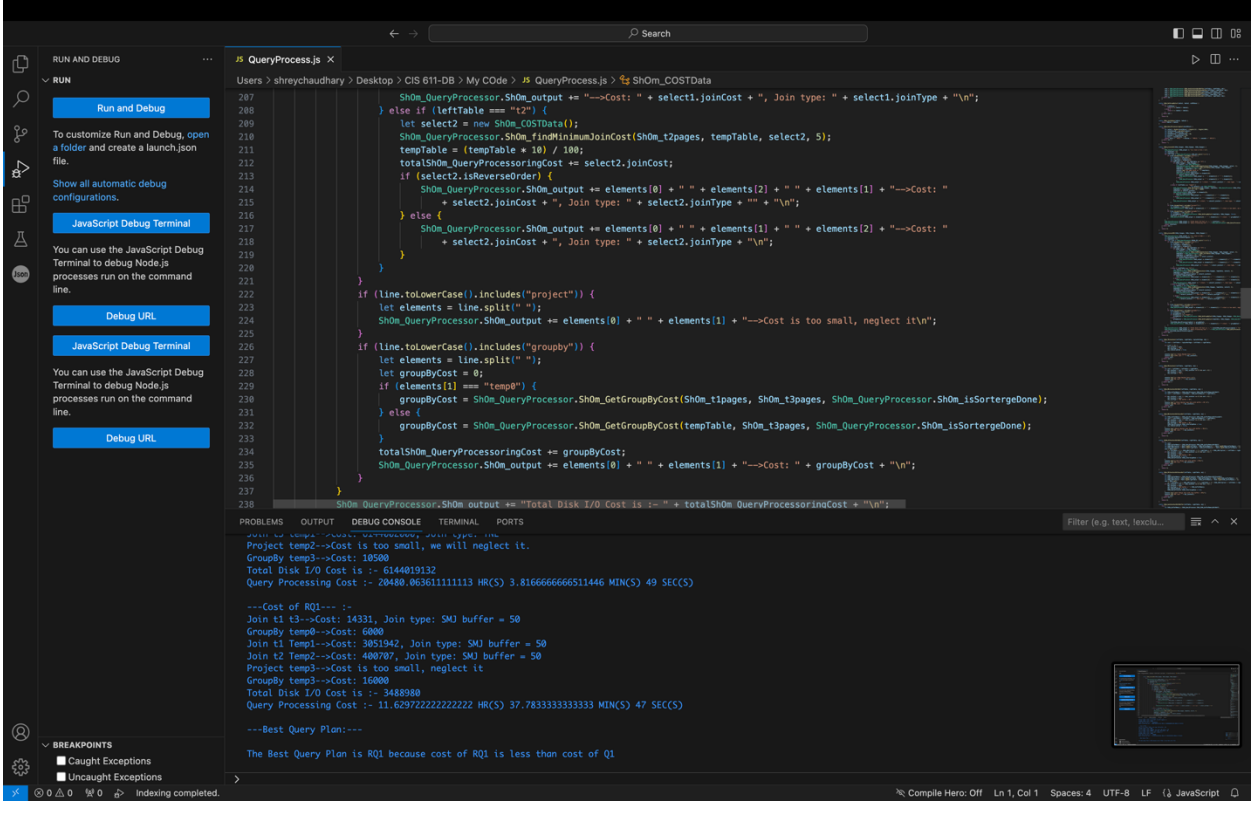

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
static ShOm_findMinimumJoinCost(leftTable, rightTable, obj, type) {
try {
let cost = 0;
if (type === 1)
cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable, ShOm_QueryProcessor.ShOm_t1TupPerPage, obj);
else if (type === 2)
cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable, ShOm_QueryProcessor.ShOm_t3TupPerPage, obj);
else if (type === 3)
cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable, ShOm_QueryProcessor.ShOm_t1TupPerPage, obj);
else if (type === 4)
cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable, ShOm_QueryProcessor.ShOm_t1TupPerPage, obj);
else if (type === 5)
cost = ShOm_QueryProcessor.ShOm_TNLJoinCost(leftTable, rightTable, ShOm_QueryProcessor.ShOm_t2TupPerPage, obj);
cost = ShOm_QueryProcessor.ShOm_PNLJoinCost(leftTable, rightTable, obj);
cost = ShOm_QueryProcessor.ShOm_hashJoinCostWithBuffer(leftTable, rightTable, obj);
cost = ShOm_QueryProcessor.ShOm_hashJoinCostWithLessBuffer(leftTable, rightTable, obj);
cost = ShOm_QueryProcessor.SShOm_MJJoinCostWithBuf(leftTable, rightTable, obj);
cost = ShOm_QueryProcessor.ShOm_SMJJoinCostWithLessBuf(leftTable, rightTable, obj);
cost = ShOm_QueryProcessor.ShOm_BNLJoinCostWithBuf(leftTable, rightTable, obj);
cost = ShOm_QueryProcessor.ShOm_BNLJoinCostWithLessBuf(leftTable, rightTable, obj);
} catch (ex) {
}
}
static ShOm_GetGroupByCost(table1, table2, isSMJDone) {
try {
if (isSMJDone) {
return 2 * (table1 + table2);
} else {
return 3 * (table1 + table2);
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, !exclu...
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

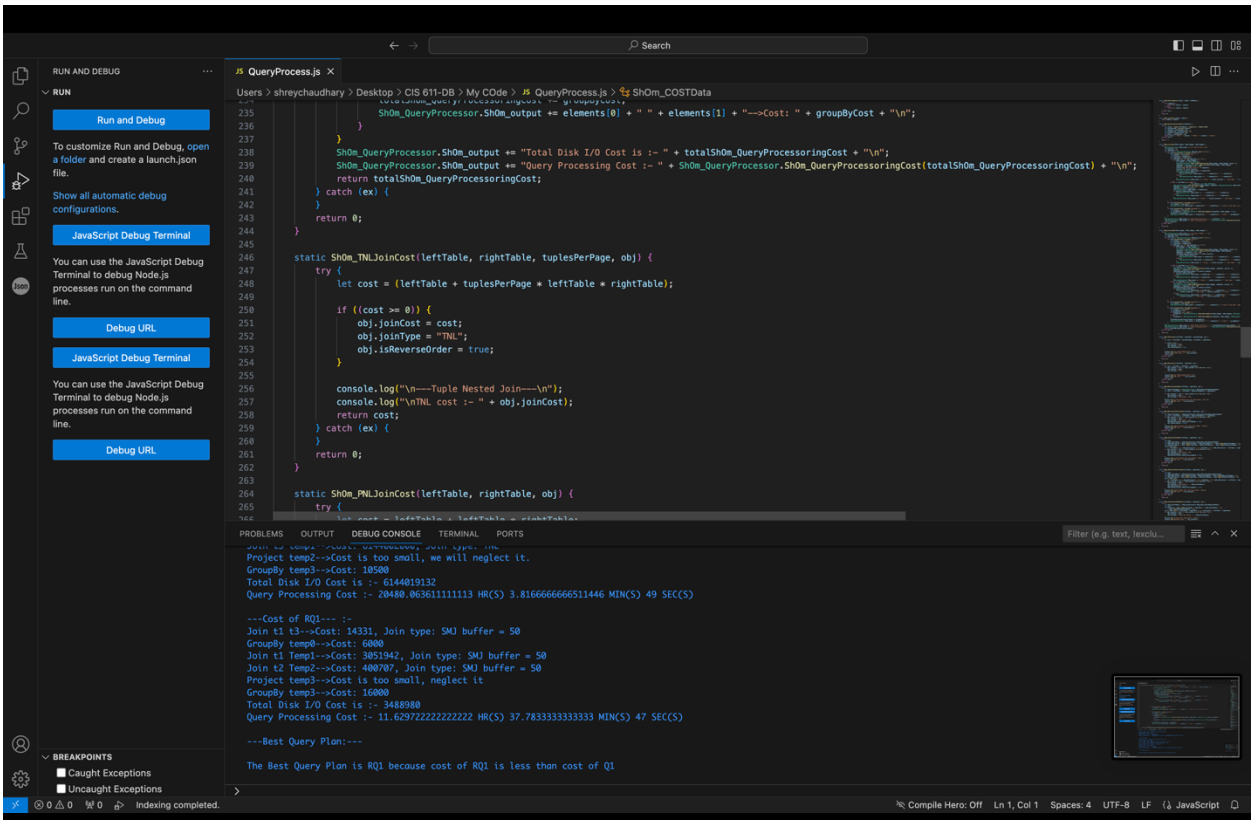

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
static ShOm_GetGroupByCost(table1, table2, isSMJDone) {
try {
if (isSMJDone) {
return 2 * (table1 + table2);
} else {
return 3 * (table1 + table2);
}
} catch (ex) {
}
return 0;
}
static ShOm_joinTables(table1, table2) {
return table1 * table2;
}
static ShOm_QueryProcessoringCost(diskIOCost) {
try {
let ioCost = BigInt(diskIOCost) * BigInt(12) / BigInt(1000);
let totalSeconds = parseInt(ioCost);
let seconds = totalSeconds % 60;
let totalMinutes = totalSeconds / 60;
let minutes = totalMinutes % 60;
let hours = totalMinutes / 60;
return hours + " HR(S) " + minutes + " MIN(S) " + seconds + " SEC(S)";
} catch (ex) {
}
return "";
}
static ShOm_processQ1(ShOm_t1pages, ShOm_t2pages, ShOm_t3pages) {
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, !exclu...
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

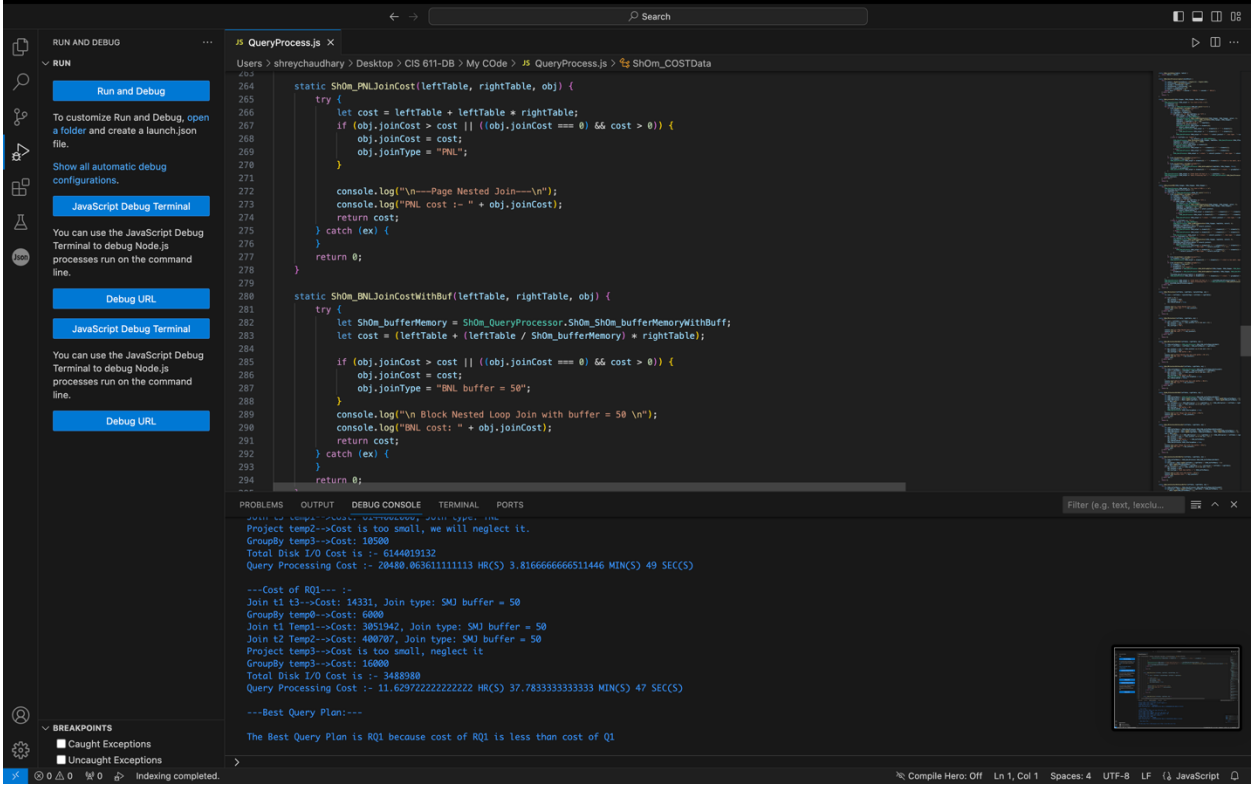

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
try {
ShOm_QueryProcessor.ShOm_output += "\n---Cost of Q1:---\n";
let totalCost = 0;
let tempTable = 0;
for (let line of ShOm_QueryProcessor.ShOm_Q1t.split("\r\n")) {
if (line.toLowerCase().includes("join")) {
let elements = line.split(" ");
let leftTable = elements[1];
let rightTable = elements[2];
if (leftTable === "t1" && rightTable === "t2") {
if (ShOm_t1pages > ShOm_t2pages) {
let select = new ShOm_COSTData();
ShOm_QueryProcessor.ShOm_findMinimumJoinCost(ShOm_t2pages, ShOm_t1pages, select, 1);
tempTable = ShOm_QueryProcessor.ShOm_joinTables(ShOm_t1pages, ShOm_t2pages);
tempTable = (tempTable * 15) / 100;
console.log("The temp value is " + tempTable);
totalCost += select.joinCost;
ShOm_QueryProcessor.ShOm_output += elements[0];
if (select.isReverseOrder) {
ShOm_QueryProcessor.ShOm_output += " " + elements[2] + " " + elements[1];
} else {
ShOm_QueryProcessor.ShOm_output += " " + elements[1] + " " + elements[2];
}
ShOm_QueryProcessor.ShOm_output += "-->Cost: " + select.joinCost + ", Join type: " + select.joinType + "\n";
}
} else if (leftTable === "temp1") {
let select2 = new ShOm_COSTData();
ShOm_QueryProcessor.ShOm_TNLJoinCost(ShOm_t3pages, tempTable, ShOm_QueryProcessor.ShOm_t3TupPerPage, select2);
tempTable = (tempTable * 10 * 20) / 10000;
totalCost += select2.joinCost;
ShOm_QueryProcessor.ShOm_output += elements[0];
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

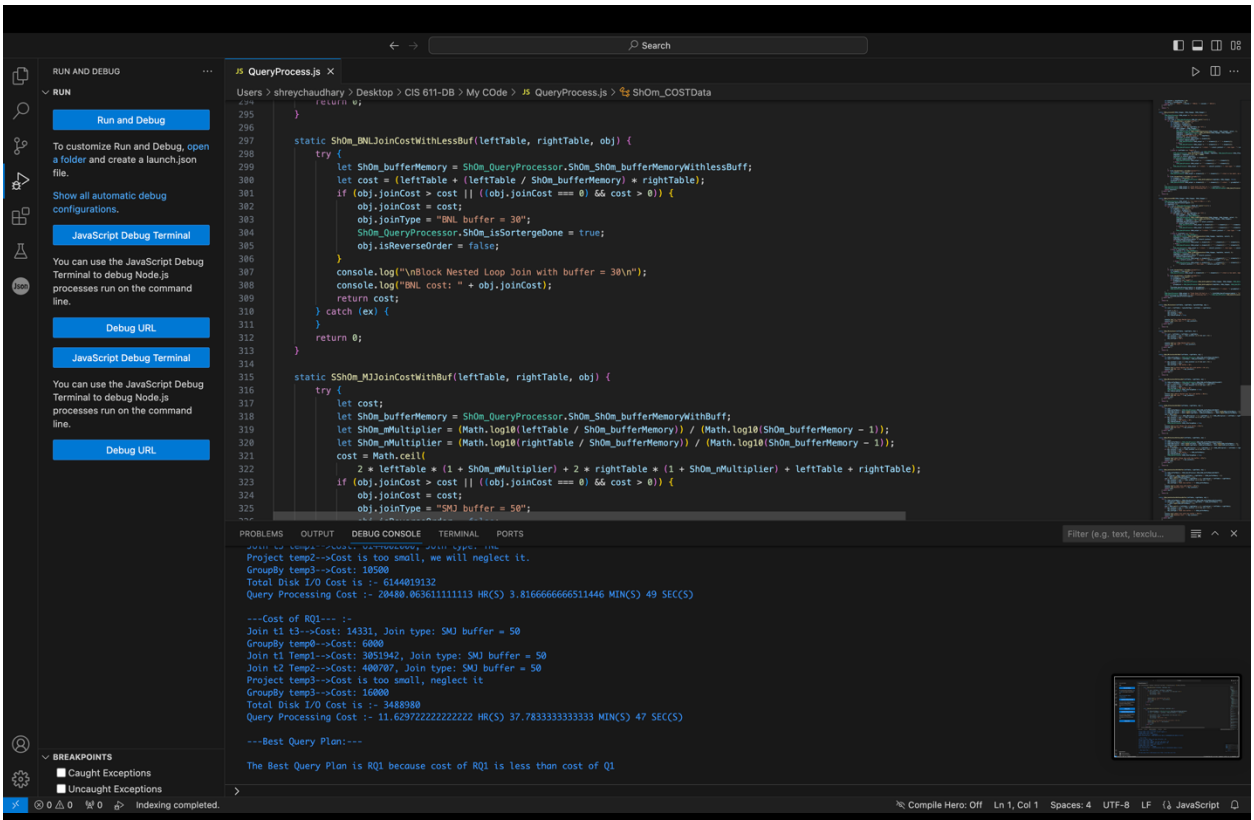

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
totalCost += select2.joinCost;
ShOm_QueryProcessor.ShOm_output += elements[0];
if (select2.isReverseOrder) {
ShOm_QueryProcessor.ShOm_output += " " + elements[2] + " " + elements[1];
} else {
ShOm_QueryProcessor.ShOm_output += " " + elements[1] + " " + elements[2];
}
ShOm_QueryProcessor.ShOm_output += "-->Cost: " + select2.joinCost + ", Join type: " + select2.joinType + "\n";
}
}
if (line.toLowerCase().includes("project")) {
let elements = line.split(" ");
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "-->Cost is too small, we will neglect it.\n";
}
if (line.toLowerCase().includes("groupby")) {
let elements = line.split(" ");
let groupByCost = ShOm_QueryProcessor.ShOm_GetGroupByCost(tempTable, ShOm_t3pages, false);
totalCost += groupByCost;
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "-->Cost: " + groupByCost + "\n";
}
}
ShOm_QueryProcessor.ShOm_output += "Total Disk I/O Cost is :- " + totalCost + "\n";
ShOm_QueryProcessor.ShOm_output += "Query Processing Cost :- " + ShOm_QueryProcessor.ShOm_QueryProcessoringCost(totalCost) + "\n";
return totalCost;
} catch (ex) {
}
return 0;
}
static ShOm_processRQ1(ShOm_t1pages, ShOm_t2pages, ShOm_t3pages) {
try {
ShOm_QueryProcessor.ShOm_output += "\n---Cost of RQ1--- :- \n";
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

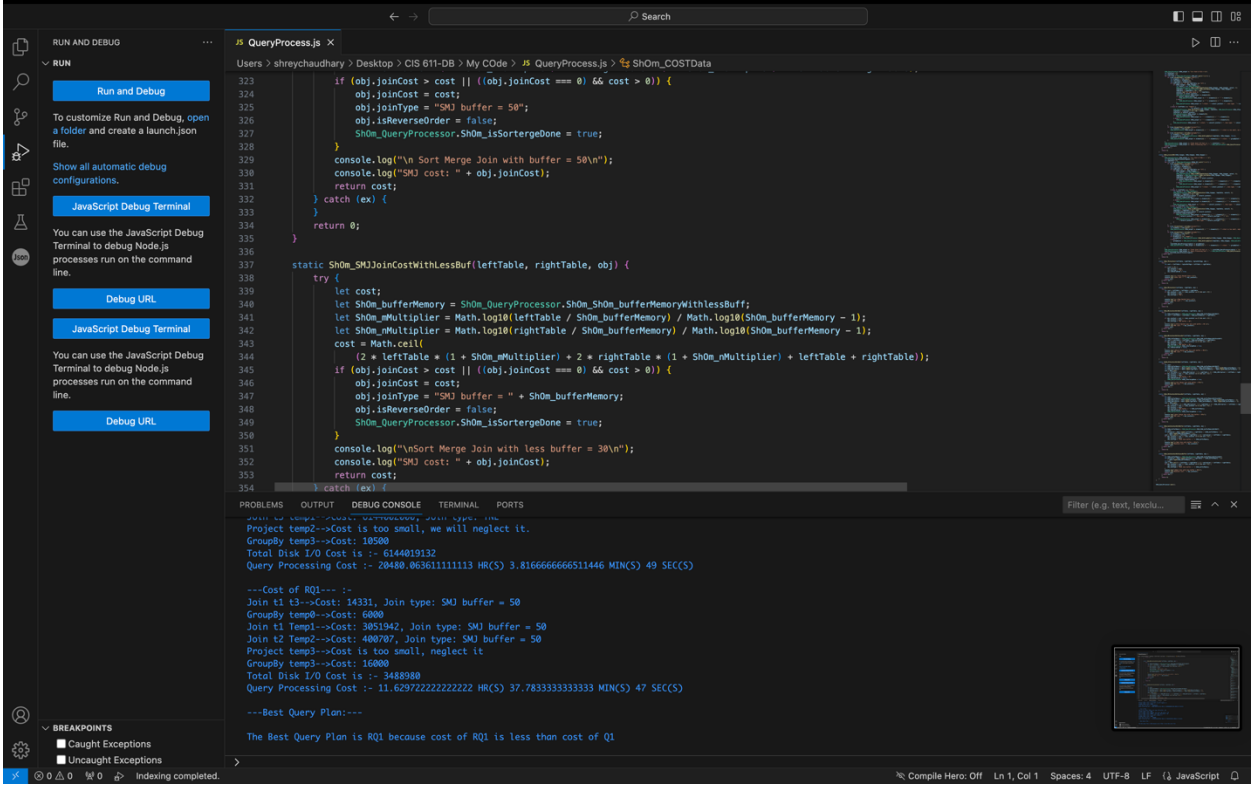

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
static ShOm_processRQ1(ShOm_t1pages, ShOm_t2pages, ShOm_t3pages) {
try {
ShOm_QueryProcessor.ShOm_output += "\n---Cost of RQ1--- :- \n";
let totalShOm_QueryProcessoringCost = 0;
let tempTable = 0;
for (let line of ShOm_QueryProcessor.RShOm_Q1t.split("\r\n")) {
if (line.toLowerCase().includes("join")) {
let elements = line.split(" ");
let leftTable = elements[1];
let rightTable = elements[2];
if (leftTable === "t1" && rightTable === "t3") {
if (ShOm_t3pages > ShOm_t1pages) {
let select = new ShOm_COSTData();
ShOm_QueryProcessor.ShOm_findMinimumJoinCost(ShOm_t3pages, ShOm_t1pages, select, 3);
tempTable = ShOm_QueryProcessor.ShOm_joinTables(ShOm_t3pages, ShOm_t1pages);
tempTable = (tempTable * 20) / 100;
totalShOm_QueryProcessoringCost += select.joinCost;
if (select.isReverseOrder) {
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[2] + " " + elements[1];
} else {
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + " " + elements[2];
}
ShOm_QueryProcessor.ShOm_output += "-->Cost: " + select.joinCost + ", Join type: " + select.joinType + "\n";
}
} else if (leftTable === "t1") {
let select1 = new ShOm_COSTData();
ShOm_QueryProcessor.ShOm_findMinimumJoinCost(ShOm_t1pages, tempTable, select1, 4);
tempTable = (tempTable * 15) / 100;
totalShOm_QueryProcessoringCost += select1.joinCost;
if (select1.isReverseOrder) {
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Filter (e.g. text, !exclu...
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

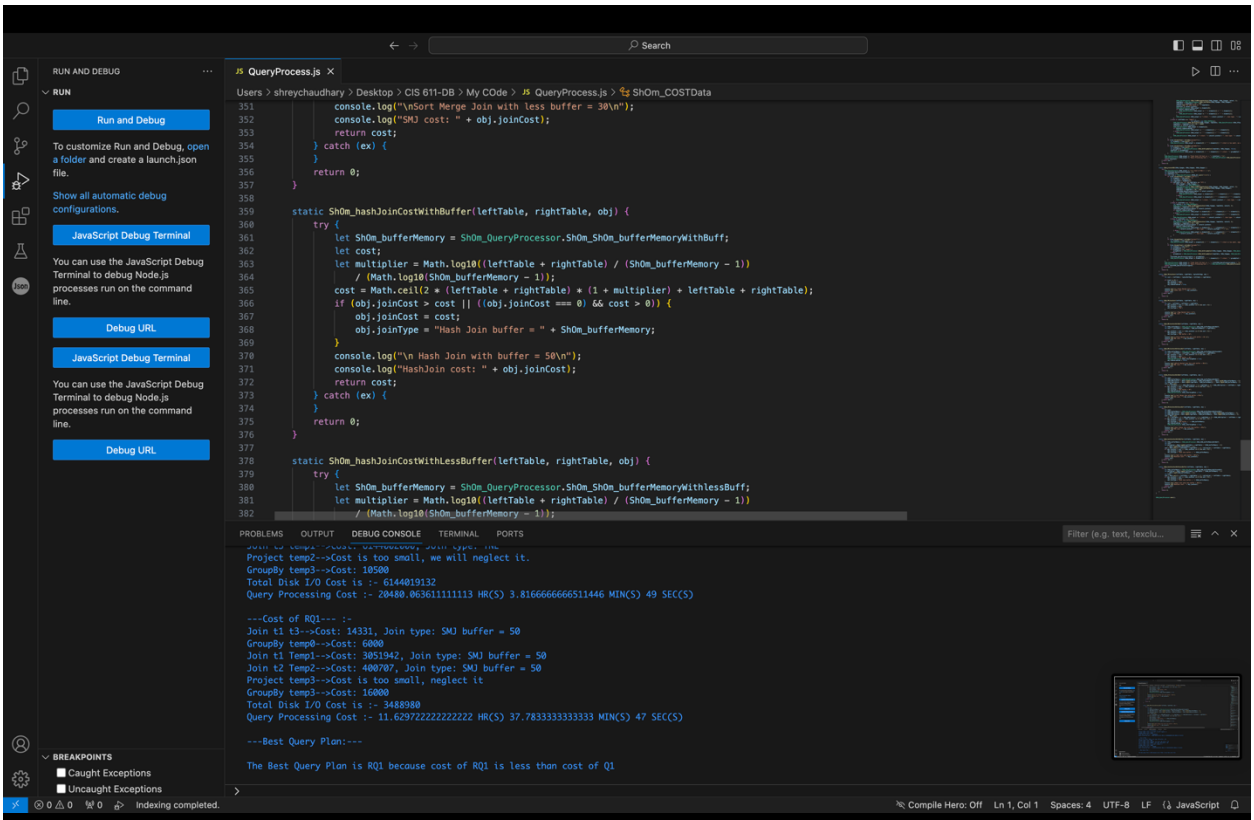

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
ShOm_QueryProcessor.ShOm_output += "-->Cost: " + select1.joinCost + ", Join type: " + select1.joinType + "\n";
} else if (leftTable === "t2") {
let select2 = new ShOm_COSTData();
ShOm_QueryProcessor.ShOm_findMinimumJoinCost(ShOm_t2pages, tempTable, select2, 5);
tempTable = (tempTable * 10) / 100;
totalShOm_QueryProcessoringCost += select2.joinCost;
if (select2.isReverseOrder) {
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[2] + " " + elements[1] + "-->Cost: "
+ select2.joinCost + ", Join type: " + select2.joinType + "" + "\n";
} else {
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + " " + elements[2] + "-->Cost: "
+ select2.joinCost + ", Join type: " + select2.joinType + "\n";
}
}
}
if (line.toLowerCase().includes("project")) {
let elements = line.split(" ");
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "-->Cost is too small, neglect it\n";
}
if (line.toLowerCase().includes("groupby")) {
let elements = line.split(" ");
let groupByCost = 0;
if (elements[1] === "temp0") {
groupByCost = ShOm_QueryProcessor.ShOm_GetGroupByCost(ShOm_t1pages, ShOm_t3pages, ShOm_QueryProcessor.ShOm_isSortergeDone);
} else {
groupByCost = ShOm_QueryProcessor.ShOm_GetGroupByCost(tempTable, ShOm_t3pages, ShOm_QueryProcessor.ShOm_isSortergeDone);
}
totalShOm_QueryProcessoringCost += groupByCost;
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "-->Cost: " + groupByCost + "\n";
}
}
ShOm_QueryProcessor.ShOm_output += "Total Disk I/O Cost is :- " + totalShOm_QueryProcessoringCost + "\n";
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Filter (e.g. text, !exclu...
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

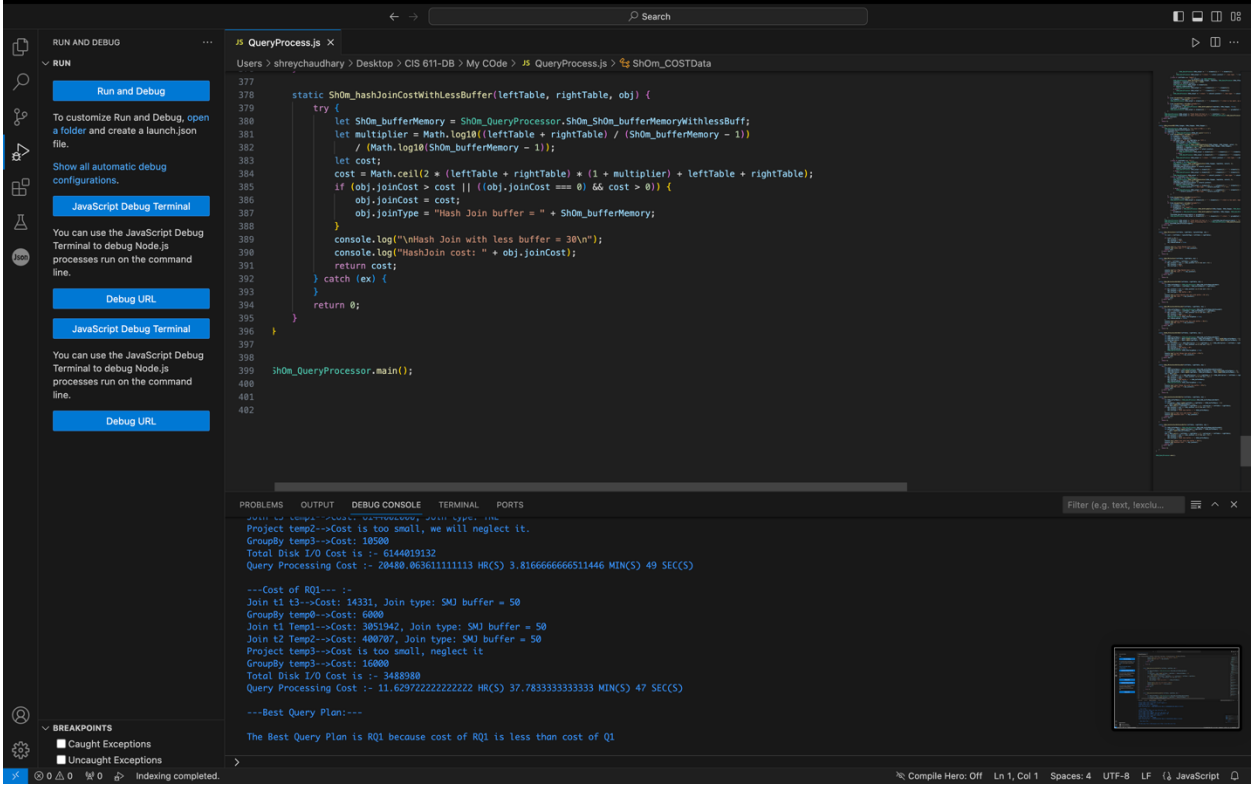

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
ShOm_QueryProcessor.ShOm_output += elements[0] + " " + elements[1] + "-->Cost: " + groupByCost + "\n";
ShOm_QueryProcessor.ShOm_output += "Total Disk I/O Cost is :- " + totalShOm_QueryProcessoringCost + "\n";
ShOm_QueryProcessor.ShOm_output += "Query Processing Cost :- " + ShOm_QueryProcessor.ShOm_QueryProcessoringCost(totalShOm_QueryProcessoringCost) + "\n";
return totalShOm_QueryProcessoringCost;
static ShOm_TNLJoinCost(leftTable, rightTable, tuplesPerPage, obj) {
let cost = (leftTable + tuplesPerPage * leftTable * rightTable);
if ((cost >= 0)) {
obj.joinCost = cost;
obj.joinType = "TNL";
obj.isReverseOrder = true;
console.log("\n---Tuple Nested Join---\n");
console.log("\nTNL cost :- " + obj.joinCost);
return cost;
static ShOm_PNLJoinCost(leftTable, rightTable, obj) {
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, !exclu...
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript


RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
static ShOm_PNLJoinCost(leftTable, rightTable, obj) {
let cost = leftTable + leftTable * rightTable;
if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
obj.joinCost = cost;
obj.joinType = "PNL";
console.log("\n---Page Nested Join---\n");
console.log("PNL cost :- " + obj.joinCost);
return cost;
static ShOm_BNLJoinCostWithBuf(leftTable, rightTable, obj) {
let ShOm_bufferMemory = ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithBuff;
let cost = (leftTable + (leftTable / ShOm_bufferMemory) * rightTable);
if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
obj.joinCost = cost;
obj.joinType = "BNL buffer = 50";
console.log("\n Block Nested Loop Join with buffer = 50 \n");
console.log("BNL cost: " + obj.joinCost);
return cost;
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, !exclu...
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

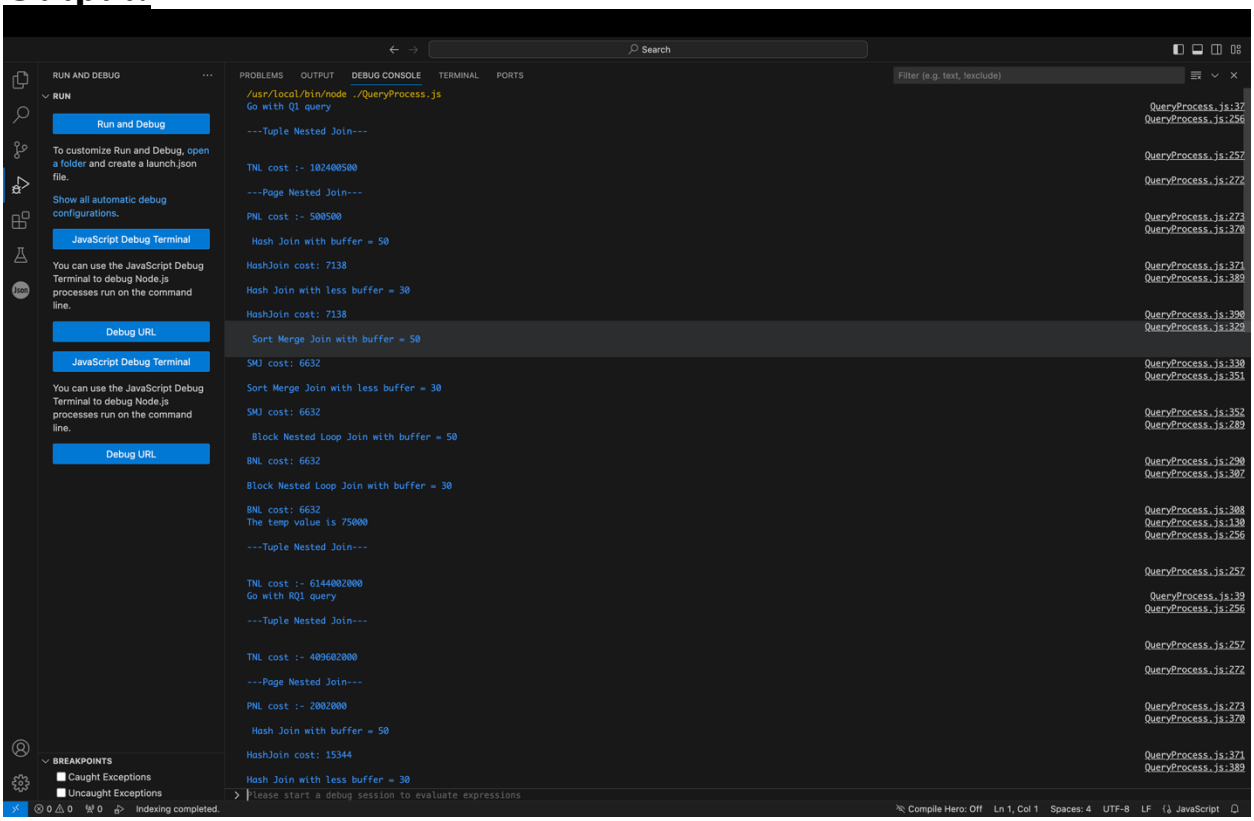

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

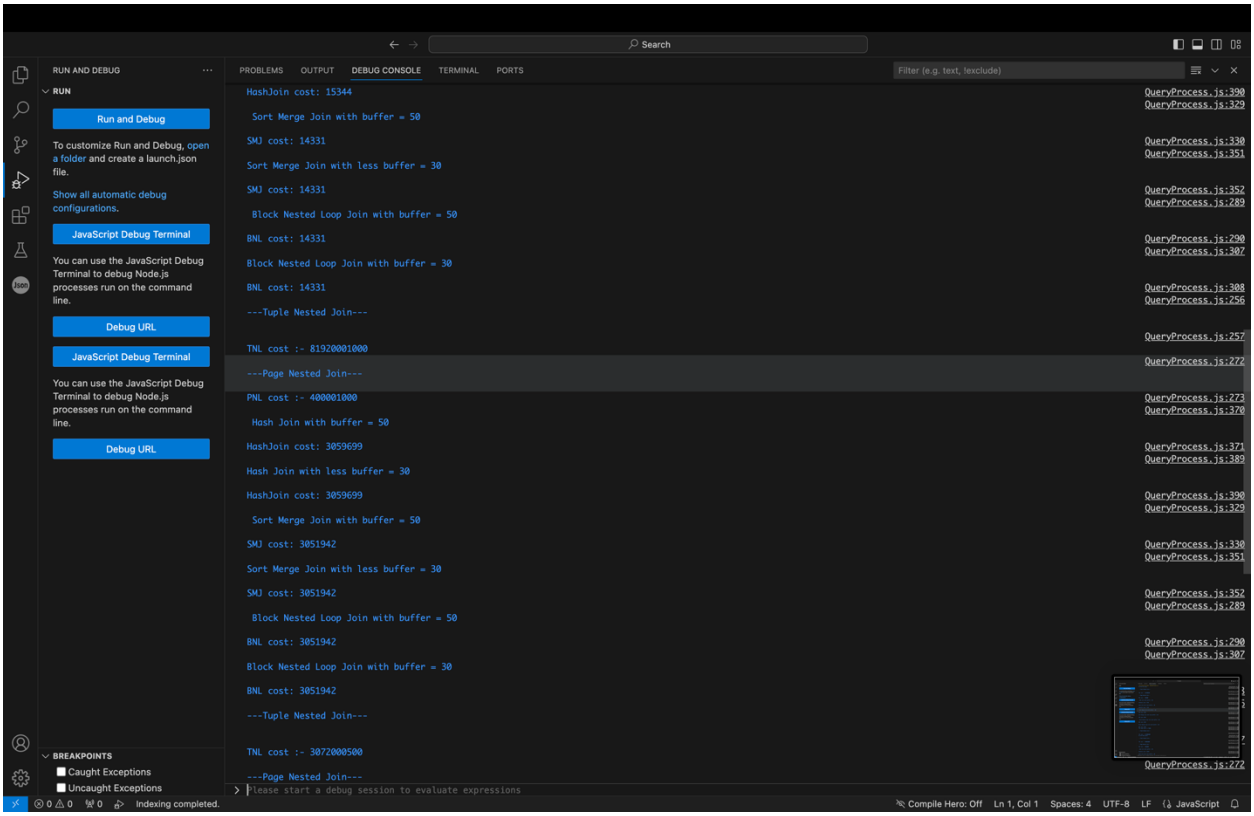

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

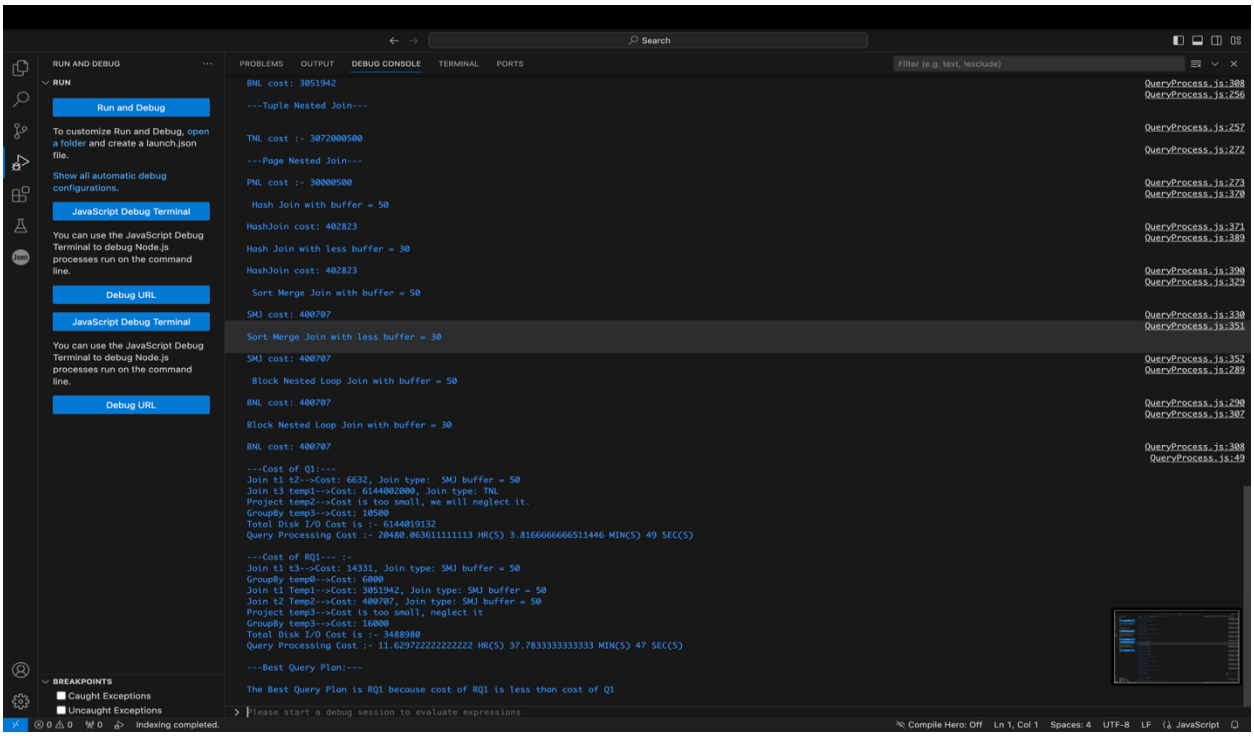

RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
console.log("\nSort Merge Join with less buffer = 30\n");
console.log("SMJ cost: " + obj.joinCost);
return cost;
} catch (ex) {
}
return 0;
}
static ShOm_hashJoinCostWithBuffer(leftTable, rightTable, obj) {
try {
let ShOm_bufferMemory = ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithBuff;
let cost;
let multiplier = Math.log10((leftTable + rightTable) / (ShOm_bufferMemory - 1))
/ (Math.log10(ShOm_bufferMemory - 1));
cost = Math.ceil(2 * (leftTable + rightTable) * (1 + multiplier) + leftTable + rightTable);
if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
obj.joinCost = cost;
obj.joinType = "Hash Join buffer = " + ShOm_bufferMemory;
}
console.log("\n Hash Join with buffer = 50\n");
console.log("HashJoin cost: " + obj.joinCost);
return cost;
} catch (ex) {
}
return 0;
}
static ShOm_hashJoinCostWithLessBuffer(leftTable, rightTable, obj) {
try {
let ShOm_bufferMemory = ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithlessBuff;
let multiplier = Math.log10((leftTable + rightTable) / (ShOm_bufferMemory - 1))
/ (Math.log10(ShOm_bufferMemory - 1));
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Filter (e.g. text, !exclu...
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript


RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
QueryProcess.js
Users > shreychaudhary > Desktop > CIS 611-DB > My COde > QueryProcess.js > ShOm_COSTData
static ShOm_hashJoinCostWithLessBuffer(leftTable, rightTable, obj) {
try {
let ShOm_bufferMemory = ShOm_QueryProcessor.ShOm_ShOm_bufferMemoryWithlessBuff;
let multiplier = Math.log10((leftTable + rightTable) / (ShOm_bufferMemory - 1))
/ (Math.log10(ShOm_bufferMemory - 1));
let cost;
cost = Math.ceil(2 * (leftTable + rightTable) * (1 + multiplier) + leftTable + rightTable);
if (obj.joinCost > cost || ((obj.joinCost === 0) && cost > 0)) {
obj.joinCost = cost;
obj.joinType = "Hash Join buffer = " + ShOm_bufferMemory;
}
console.log("\nHash Join with less buffer = 30\n");
console.log("HashJoin cost: " + obj.joinCost);
return cost;
} catch (ex) {
}
return 0;
}
}
ShOm_QueryProcessor.main();
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Filter (e.g. text, !exclu...
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript

Output:


RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
/usr/local/bin/node ./QueryProcess.js
Go with Q1 query
---Tuple Nested Join---
TNL cost :- 102400500
---Page Nested Join---
PNL cost :- 500500
Hash Join with buffer = 50
HashJoin cost: 7138
Hash Join with less buffer = 30
HashJoin cost: 7138
Sort Merge Join with buffer = 50
SMJ cost: 6632
Sort Merge Join with less buffer = 30
SMJ cost: 6632
Block Nested Loop Join with buffer = 50
BNL cost: 6632
Block Nested Loop Join with buffer = 30
BNL cost: 6632
The temp value is 75000
---Tuple Nested Join---
TNL cost :- 6144002000
Go with RQ1 query
---Tuple Nested Join---
TNL cost :- 409602000
---Page Nested Join---
PNL cost :- 2002000
Hash Join with buffer = 50
HashJoin cost: 15344
Hash Join with less buffer = 30
Please start a debug session to evaluate expressions
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript


RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
HashJoin cost: 15344
Sort Merge Join with buffer = 50
SMJ cost: 14331
Sort Merge Join with less buffer = 30
SMJ cost: 14331
Block Nested Loop Join with buffer = 50
BNL cost: 14331
Block Nested Loop Join with buffer = 30
BNL cost: 14331
---Tuple Nested Join---
TNL cost :- 81920001000
---Page Nested Join---
PNL cost :- 400001000
Hash Join with buffer = 50
HashJoin cost: 3059699
Hash Join with less buffer = 30
HashJoin cost: 3059699
Sort Merge Join with buffer = 50
SMJ cost: 3051942
Sort Merge Join with less buffer = 30
SMJ cost: 3051942
Block Nested Loop Join with buffer = 50
BNL cost: 3051942
Block Nested Loop Join with buffer = 30
BNL cost: 3051942
---Tuple Nested Join---
TNL cost :- 3072000500
---Page Nested Join---
Please start a debug session to evaluate expressions
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript


RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug, open a folder and create a launch.json file.
Show all automatic debug configurations.
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
JavaScript Debug Terminal
You can use the JavaScript Debug Terminal to debug Node.js processes run on the command line.
Debug URL
BREAKPOINTS
Caught Exceptions
Uncaught Exceptions
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, !exclude)
BNL cost: 3051942
---Tuple Nested Join---
TNL cost :- 3072000500
---Page Nested Join---
PNL cost :- 30000500
Hash Join with buffer = 50
HashJoin cost: 402823
Hash Join with less buffer = 30
HashJoin cost: 402823
Sort Merge Join with buffer = 50
SMJ cost: 400707
Sort Merge Join with less buffer = 30
SMJ cost: 400707
Block Nested Loop Join with buffer = 50
BNL cost: 400707
Block Nested Loop Join with buffer = 30
BNL cost: 400707
---Cost of Q1:---
Join t1 t2-->Cost: 6632, Join type: SMJ buffer = 50
Join t3 temp1-->Cost: 6144002000, Join type: TNL
Project temp2-->Cost is too small, we will neglect it.
GroupBy temp3-->Cost: 10500
Total Disk I/O Cost is :- 6144019132
Query Processing Cost :- 20480.063611111113 HR(S) 3.8166666666511446 MIN(S) 49 SEC(S)
---Cost of RQ1--- :-
Join t1 t3-->Cost: 14331, Join type: SMJ buffer = 50
GroupBy temp0-->Cost: 6000
Join t1 Temp1-->Cost: 3051942, Join type: SMJ buffer = 50
Join t2 Temp2-->Cost: 400707, Join type: SMJ buffer = 50
Project temp3-->Cost is too small, neglect it
GroupBy temp3-->Cost: 16000
Total Disk I/O Cost is :- 3488980
Query Processing Cost :- 11.629722222222222 HR(S) 37.7833333333333 MIN(S) 47 SEC(S)
---Best Query Plan:---
The Best Query Plan is RQ1 because cost of RQ1 is less than cost of Q1
QueryProcess.js:308
QueryProcess.js:256
QueryProcess.js:257
QueryProcess.js:272
QueryProcess.js:273
QueryProcess.js:370
QueryProcess.js:371
QueryProcess.js:389
QueryProcess.js:390
QueryProcess.js:329
QueryProcess.js:330
QueryProcess.js:351
QueryProcess.js:352
QueryProcess.js:289
QueryProcess.js:290
QueryProcess.js:307
QueryProcess.js:308
QueryProcess.js:49
Please start a debug session to evaluate expressions
Indexing completed.
Compile Hero: Off
Ln 1, Col 1
Spaces: 4
UTF-8
LF
JavaScript