



COMP - 6521
Advanced Database Technology and Applications

Project Report
on
Sort Based and Nested Loop Join

Professor
Dr. Nematollaah Shiri

Team Members

Student Name	Student Id	Email Id
Sagar Vetal	40071979	s_vetal@encs.concordia.ca
Himanshu Kohli	40070839	h_kohli@encs.concordia.ca
Nayana Raj Cheluvaraju	40071318	n_cheluv@encs.concordia.ca
Balpreet Singh	40070817	s_balpre@encs.concordia.ca

Contents:

1. How to run the program	2
2. Program Description and Calculations.....	2
3. Algorithms.....	6
4. Flow Diagrams.....	7
5. Coding Standards used.....	9
6. Architecture Diagram.....	9
7. Description of the Classes	10
8. Result.....	11

1. How to run the program:

- Place the data sets in the 'inputfiles' folder and set up the eclipse environment.
- Setup main memory size to 5 MB.
- Run the program with 'JoinDataController.java'
- For sort-based join the program will read the input relations T1 & T2 from 'inputfiles' folder based on main memory size, sort them and perform join operation on the sorted relations along with student GPA calculation.
- For nested-loop join the program will read the input relations T1 & T2 from 'inputfiles' folder based on main memory size and perform join operation.
- The output (i.e. sorted relations named JoinT1_sorted.txt, JoinT2_sorted.txt, NestedLoopJoinResult.txt, SortBasedJoinResult.txt and StudentGPA.txt) will be generated in the 'outputfiles' folder.

2. Program Description and Calculations:

Sort-Based Join:

The program reads the input relations T1 & T2 from 'inputfiles' folder based on main memory size one after the other and processes through TPMMS phase 1 and phase 2. The sub-lists created in phase 1 can be seen in the 'sublists' folder and the complete sorted relations named 'JoinT1_sorted.txt' and 'JoinT2_sorted.txt' created in phase 2 can be seen in 'outputfiles' folder. Then it will read sorted relations and perform join operation along with student GPA calculation. Results of Sort-based join and Student GPA calculation (named 'SortBasedJoinResult.txt' and 'StudentGPA.txt') can be seen in 'outputfiles' folder.

Nested-Loop Join:

The program will read the input relations T1 & T2 from 'inputfiles' folder based on main memory size. As $B(T1) < B(T2)$ program will read blocks of T1 in outer loop and blocks of T2 in inner loop. It will repeatedly read M-1 blocks of T1 into main memory buffers. A search structure with search key 'student_id' is created for the tuples of T1 that are in main memory. Then it will read all the blocks of T2 one by one into the last block of memory. It will search each tuple of T2 present in main memory into a search structure of the tuples of T1 with common attribute 'student_id'. Output all the tuples that can be formed by joining tuples from T1 and T2 that have common attribute with common value. Result of nested-loop join (named 'NestedLoopJoinResult.txt') can be seen in 'outputfiles' folder.

Student GPA:

Program will calculate GPA of each student while computing sort-based join using following formula,

$$GPA = \frac{\text{Total (credit hours per course } \times \text{ grade received per course)}}{\text{Total credit hours taken during all semesters}}$$

Calculation:

The calculations for total numbers of blocks of both relations, number of blocks main memory can hold, and both join operations are shown below,

As we are holding data in bytes thought out program, calculation is done using same,

- Size of Input data T1 = 5,050,101 bytes.
- Size of Input data T2 = 153,996,836 bytes.
- Memory size = 5MB (50% of given memory size for datasets and remaining 50% for program execution i.e. 2.5MB each).

Therefore, below calculation is shown for 2.5MB

- No of Blocks main memory can hold

$$= \text{memory size} / \text{block size}$$

$$= 2.5\text{MB} / 4\text{KB}$$

$$= 640 \text{ Blocks}$$
- No of Tuples in T1

$$= \text{T1 size} / 1 \text{ tuple size}$$

$$= 5,050,101 / 101$$

$$= 50001 \text{ Tuples}$$

(Note: Here tuple size is taken 101 bytes instead of 100 bytes as there is a \n for new line for every tuple.)
- No of Blocks in T1 = B(T1)

$$= \text{Total no. of tuples} / \text{No. of tuples in 1 Block}$$

$$= 50001 / 40$$

$$= 1251 \text{ Blocks}$$
- No of Tuples in T2

$$= \text{T2 size} / 1 \text{ tuple size}$$

$$= 153,996,836 / 28$$

$$= 5499887 \text{ Tuples}$$

(Note: Here tuple size is taken 28 bytes instead of 27 bytes as there is a \n for new line for every tuple.)
- No of Blocks in T2 = B(T2)

$$= \text{Total no. of tuples} / \text{No. of tuples in 1 Block}$$

$$= 5499887 / 130$$

$$= 42307 \text{ Blocks}$$

Sort-Based Join:

- Total Disk I/O's to sort T1 & T2

$$= 4 * (B(T1) + B(T2))$$

$$= 4 * (1251 + 42307)$$

$$= 174, 232$$
- Total Disk I/O's to compute join operation on sorted T1 & T2 = 9,893

- Size of joined result = 125.94 MB (i.e. 132,055,920 bytes)
- Size of 1 joined tuple = 120 bytes

- No of tuples in joined result
 $= 132,055,920 / 120$
 $= 1,100,466$

- No of joined tuples in 1 block
 $= 4096 / 120$
 $= 34$

- Total Disk I/O's to write joined tuples
 $= \text{No. of joined tuples} / \text{No. of joined tuples in 1 Block}$
 $= 1,100,466 / 34$
 $= 32,367$

- **Total Disk I/O's for sort-based join operation**
 $= \text{Disk I/O's to sort T1 \& T2} + \text{Disk I/O's to compute join operation} + \text{Disk I/O's to write join output}$
 $= (4 * (B(T1) + B(T2))) + 9,893 + 32,367$
 $= 174,232 + 9,893 + 32,367$
 $= 216,490$

Nested-Loop Join:

Here we are allocating 30% of memory blocks for T1 buffer, 20% for T2 buffer, 10% for output buffer and remaining for a search structure.

- 30% of the memory Blocks will be used for Relation T1
 $= (30\% \text{ of No. of Block in Main Memory})$
 $= (0.30 * 640)$
 $= 192 \text{ Blocks}$

- No of iterations of the outer loop
 $= B(T1) / \text{memory blocks used for T1}$
 $= 1251 / 192$
 $= 7$

- Total Disk I/O's to compute join operation
 $= B(T1) + (B(T1) / \text{memory blocks used for T1}) * B(T2)$
 $= 1251 + (1251 / 192) * 42,307$
 $= 1251 + 7 * 42,307$
 $= 297,400$

- Size of joined result = 125.94 MB (i.e. 132,055,920 bytes)
- Size of 1 joined tuple = 120 bytes

- No of tuples in joined result
 $= 132,055,920 / 120$
 $= 1,100,466$

- No of joined tuples in 1 block
 $= 4096 / 120$
 $= 34$
- Total Disk I/O's to write joined tuples
 $= \text{No. of joined tuples} / \text{No. of joined tuples in 1 Block}$
 $= 1,100,466 / 34$
 $= 32,367$
- **Total Disk I/O's for Nested-Loop join operation**
 $= \text{Disk I/O's to compute join operation} + \text{Disk I/O's to write joined tuples}$
 $= 297,400 + 32,367$
 $= 329,767$

Summary:

Join Method	Disk I/O's	Execution Time	Output File Size
Sort-Based Join	216,490	11.93 Seconds	125.94 MB
Nested-Loop Join	329,767	38.27 Seconds	125.94 MB

The program calculates the following which are displayed in console:

- Main Memory Size.
- Available main memory for Data sets
- Total main memory blocks
- Total number of tuples in T1.
- Total number of blocks in T1.
- Total number of tuples in T2.
- Total number of blocks in T2.
- Tuples in Join Result (Sort-Based Join)
- Join Result Size (Sort-Based Join)
- Total Disk I/O (Sort-Based Join)
- Total Execution Time (Sort-Based Join)
- Tuples in Join Result (Nested-Loop Join)
- Join Result Size (Nested-Loop Join)
- Total Disk I/O (Nested-Loop Join)
- Total Execution Time (Nested-Loop Join)

3. Algorithms:

Here are the algorithms for Sort-Based join and Nested-Loop join.

Sort-Based Join:

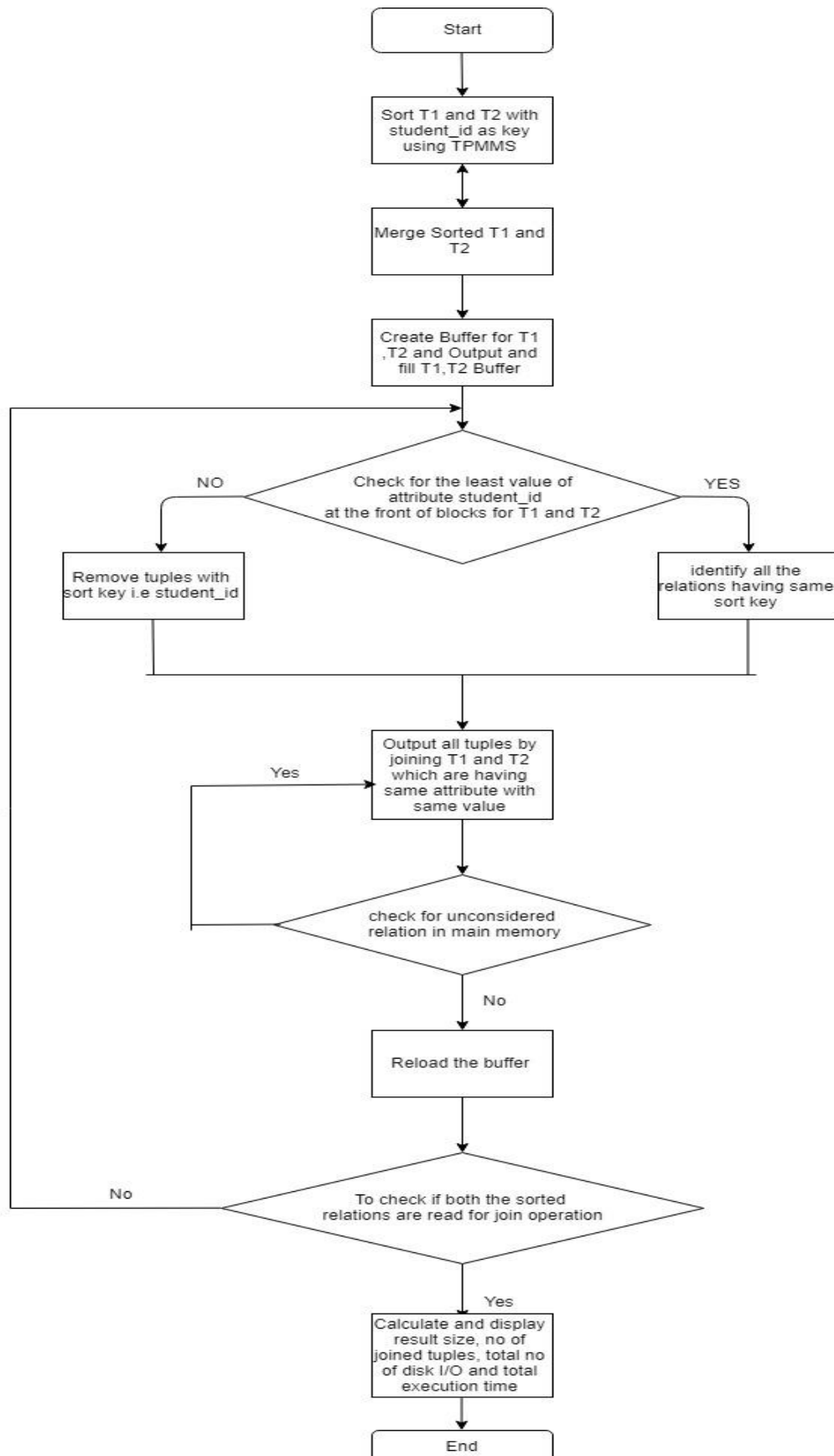
1. Start.
2. Sort T1 using TPMMS, with 'student_id' as the sort key.
3. Sort T2 similarly.
4. Merge the sorted T1 and T2.
5. Create three buffers: one is for the blocks of T1, second one is the blocks of T2 and last one is for the output.
6. Fill the buffer of T1 and T2 into main memory.
7. Find the least value of the join attribute 'student_id' that is currently at the front of the blocks for T1 and T2.
8. If the least value does not appear at the front of the other relation, then remove the tuples with sort key.
9. Otherwise, identify all the tuples from both relations having same sort key.
10. Output all the tuples that can be formed by joining tuples from T1 and T2 that have common attribute with common value.
11. Reload the buffers if either relation has no more unconsidered tuples in main memory.
12. Repeat step 7 to 11 until both sorted relations are read for join operation.
13. Calculate and display result size, the number of joined tuples, total number of Disk I/O and total execution time.
14. End.

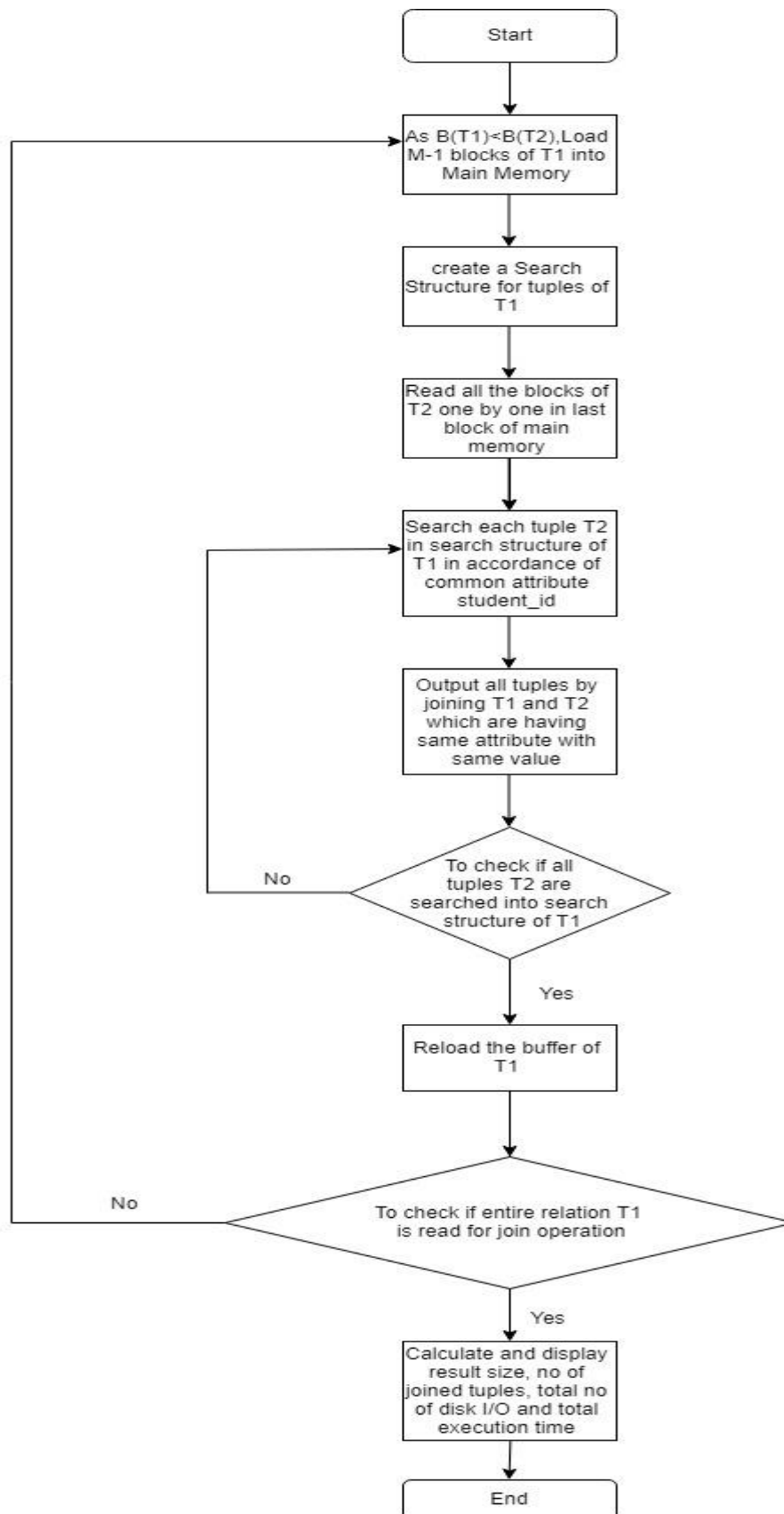
Nested Loop Join:

1. Start
2. As $B(T1) < B(T2)$, load M-1 blocks of T1 into main memory buffers.
3. Create a search structure for the tuples of T1 that are in main memory.
4. Now read all the blocks of T2 one by one into last memory block.
5. Search each tuple of T2 present in main memory into a search structure of the tuples of T1 with common attribute 'student_id'.
6. Output all the tuples that can be formed by joining tuples from T1 and T2 that have common attribute with common value.
7. Reload the buffer of T1 if all the tuples of T2 are searched into a search structure of the tuples of T1
8. Repeat step 2 to 7 until entire T1 relation is read for join operation.
9. Calculate and display result size, the number of joined tuples, total number of Disk I/O and total execution time.
10. End.

4. Flow Diagrams:

Sort-Based Join:



Nested-Loop Join:

5. Coding Standards used:

The most general coding conventions have been followed while developing the codes which are as follows,

- The name of the classes starts with a upper case character.
E.g.: JoinDataController.java
- Constants are named with upper case characters and include underscore between two words (if applicable).
- The name of the variables is descriptive and are written in lower case including a capital letter to separate between words.
- The name of the methods starts with a lower-case character and use uppercase letters to separate words.

6. Architecture Diagram:

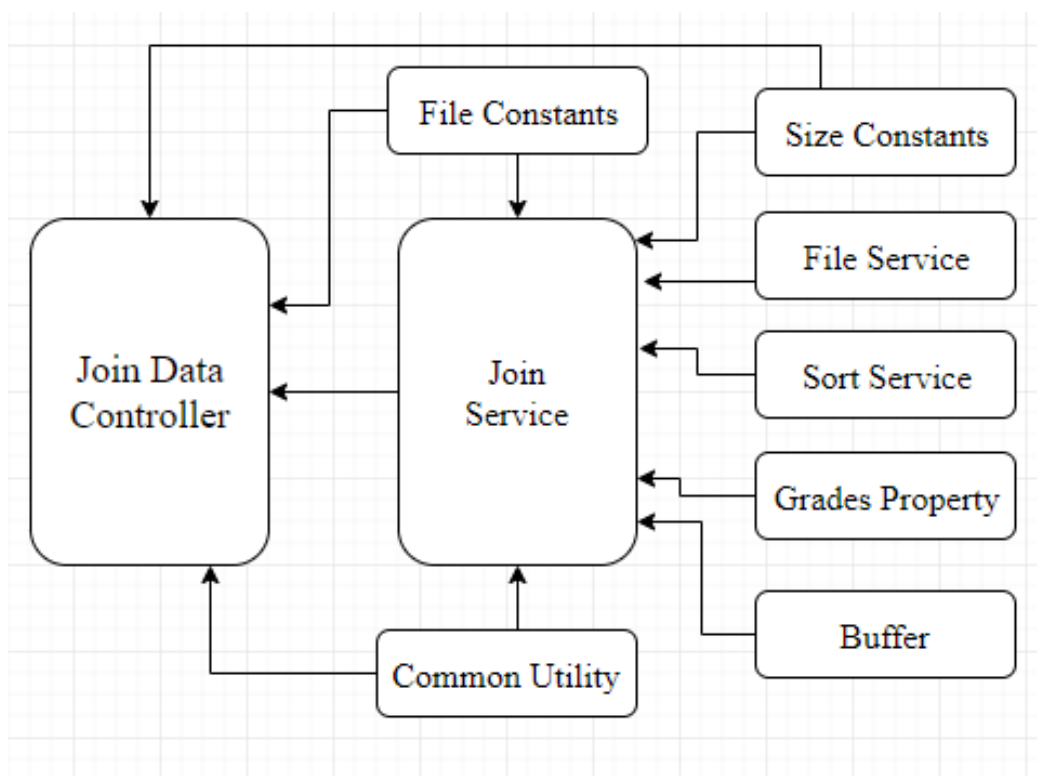


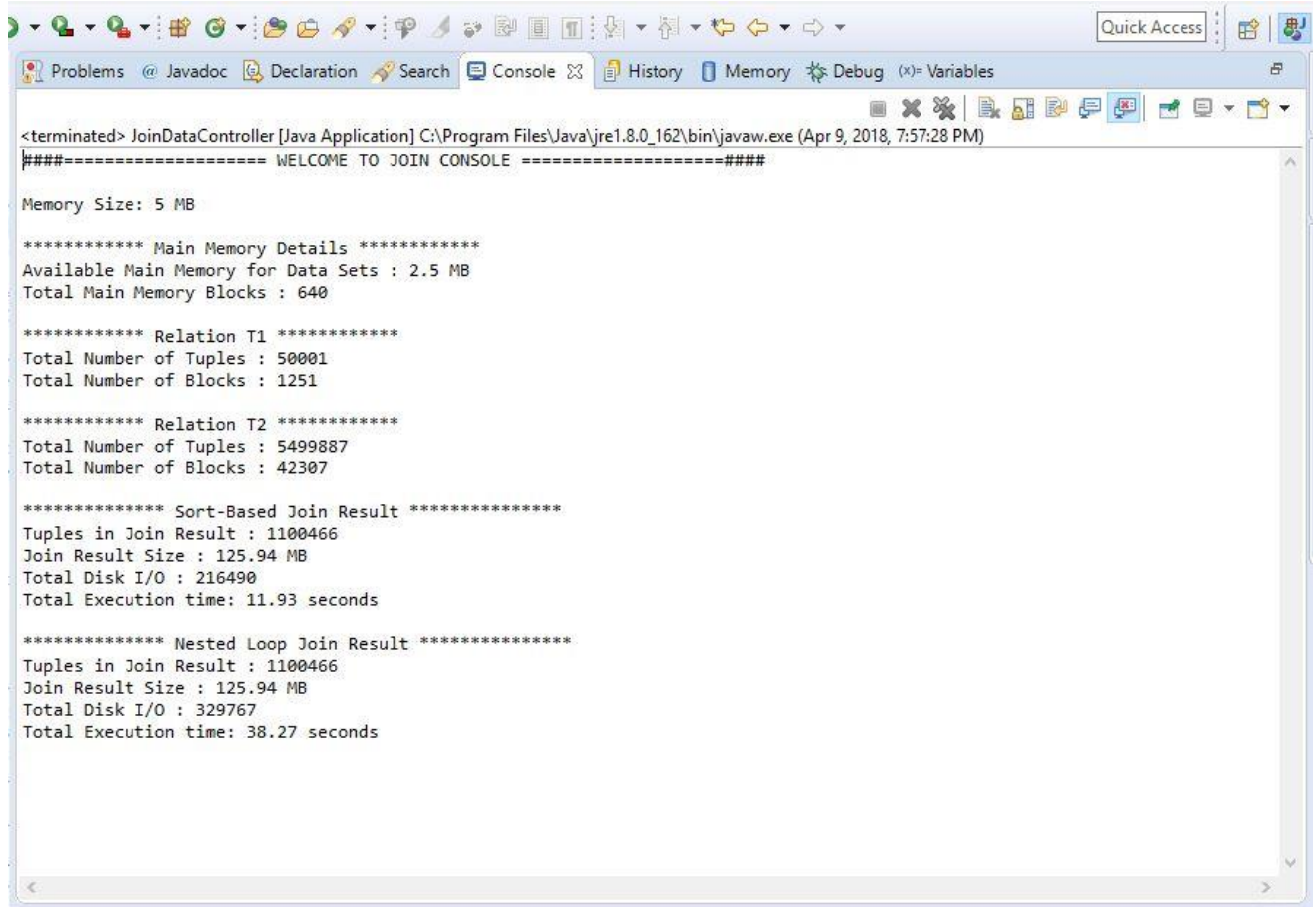
Figure: Architecture Diagram

7. Description of the Classes:

- **JoinDataController.java:**
This class is the main executable class which invokes the services for read, write, sort and join operations.
- **JoinService.java:**
This class is a service in which methods are defined for sorting, merging, nested-loop join, sort-based join operations and student GPA calculation.
- **Buffer.java:**
This class is act as a model which holds the actual data from given relations based on allocated buffer size and current and last position of the given relation. It also includes compareTo() method to compare the tuples of given relations.
- **FileService.java**
This class is used to read and write data from the given relation.
- **FileConstants.java**
This class holds all the constant values for the project such as input file names, output files names and the directory locations.
- **SizeConstatnts.java**
This class holds the constants values for main memory size, search key size, block size and tuple size.
- **SortService.java**
This class is a service in which method is defined for quick sort.
- **CommonUtility.java**
This utility class is used to perform various calculations such as total number of tuples in given relation, total number of tuple per block, total number of blocks, number of main memory blocks, number of main memory fills, main memory size in MB and round double value to 2 decimals.
- **Grades.properties**
This property file contains all grades and its equivalent credit points.

8. Results:

Sort-Based Join & Nested-Loop Join:



```

<terminated> JoinDataController [Java Application] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (Apr 9, 2018, 7:57:28 PM)
##### WELCOME TO JOIN CONSOLE #####

Memory Size: 5 MB

***** Main Memory Details *****
Available Main Memory for Data Sets : 2.5 MB
Total Main Memory Blocks : 640

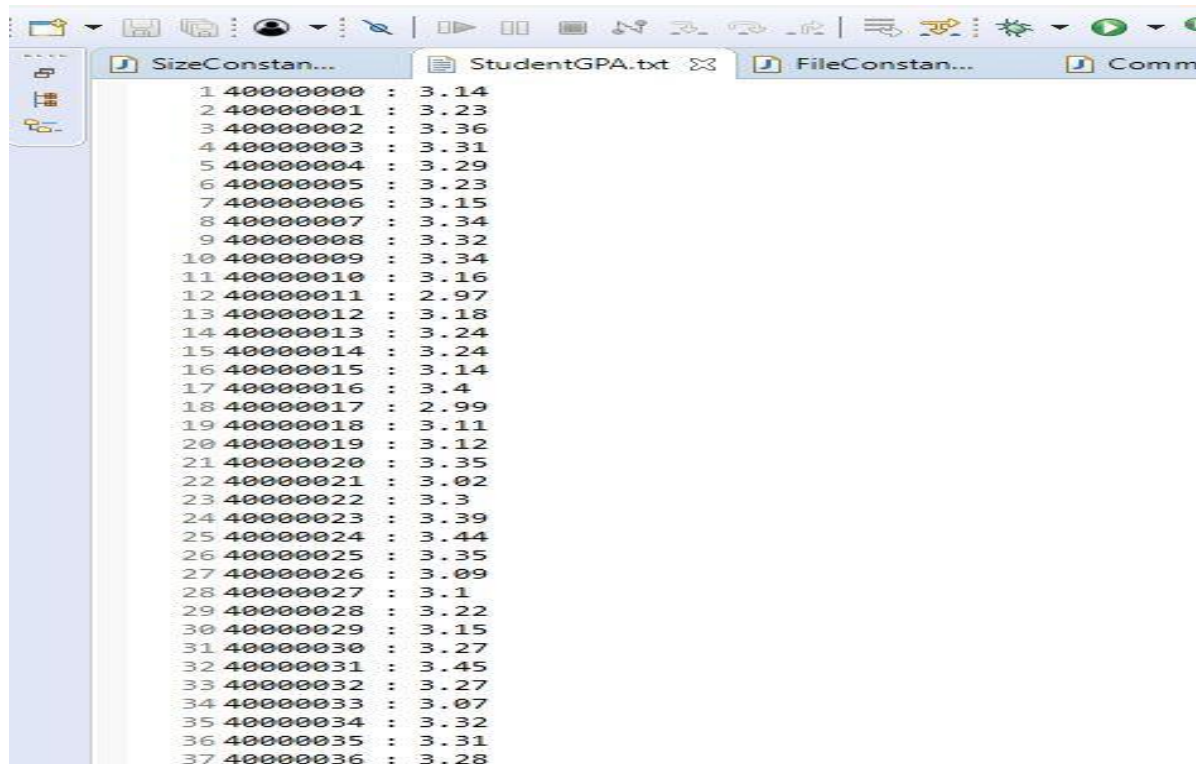
***** Relation T1 *****
Total Number of Tuples : 50001
Total Number of Blocks : 1251

***** Relation T2 *****
Total Number of Tuples : 5499887
Total Number of Blocks : 42307

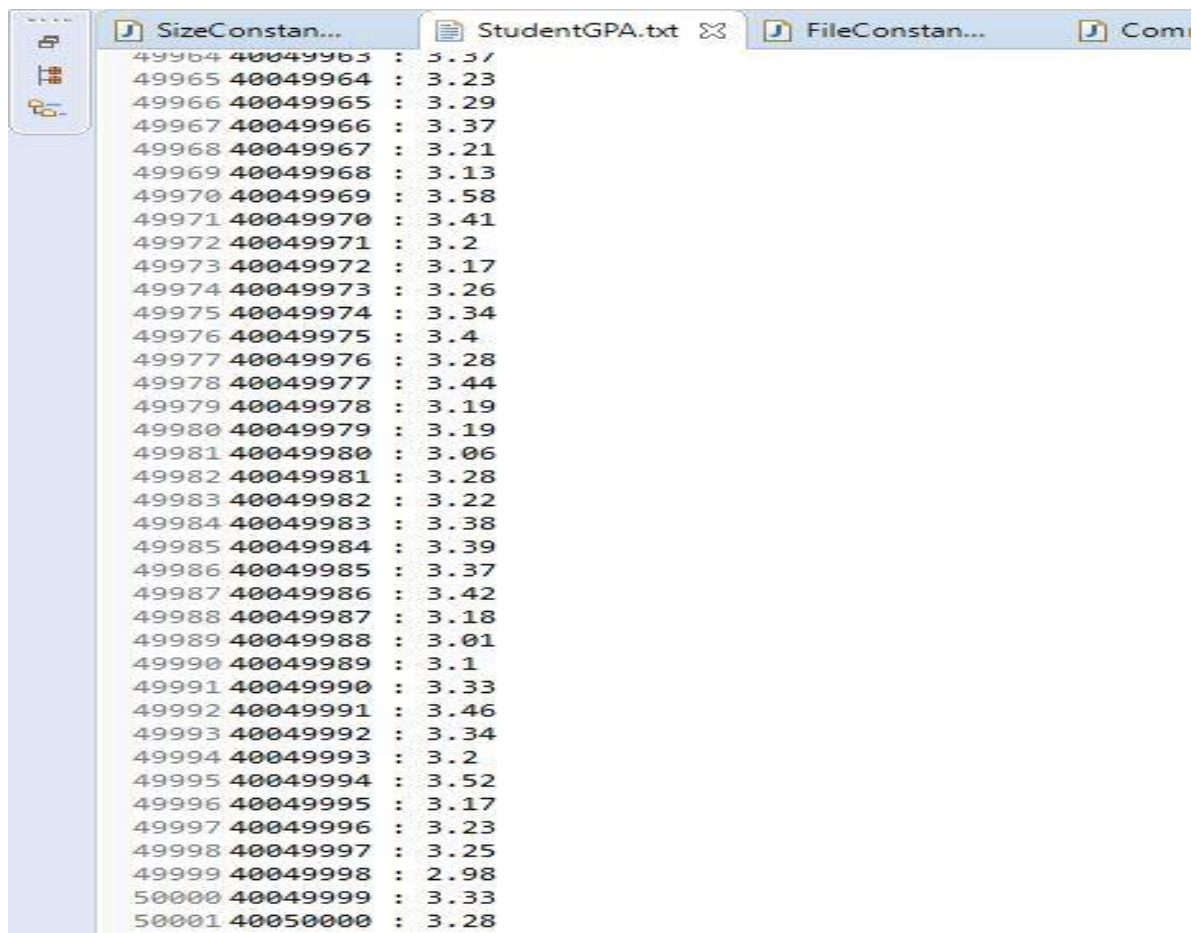
***** Sort-Based Join Result *****
Tuples in Join Result : 1100466
Join Result Size : 125.94 MB
Total Disk I/O : 216490
Total Execution time: 11.93 seconds

***** Nested Loop Join Result *****
Tuples in Join Result : 1100466
Join Result Size : 125.94 MB
Total Disk I/O : 329767
Total Execution time: 38.27 seconds
  
```

Student GPA:



1	400000000	: 3.14
2	400000001	: 3.23
3	400000002	: 3.36
4	400000003	: 3.31
5	400000004	: 3.29
6	400000005	: 3.23
7	400000006	: 3.15
8	400000007	: 3.34
9	400000008	: 3.32
10	400000009	: 3.34
11	400000010	: 3.16
12	400000011	: 2.97
13	400000012	: 3.18
14	400000013	: 3.24
15	400000014	: 3.24
16	400000015	: 3.14
17	400000016	: 3.4
18	400000017	: 2.99
19	400000018	: 3.11
20	400000019	: 3.12
21	400000020	: 3.35
22	400000021	: 3.02
23	400000022	: 3.3
24	400000023	: 3.39
25	400000024	: 3.44
26	400000025	: 3.35
27	400000026	: 3.09
28	400000027	: 3.1
29	400000028	: 3.22
30	400000029	: 3.15
31	400000030	: 3.27
32	400000031	: 3.45
33	400000032	: 3.27
34	400000033	: 3.07
35	400000034	: 3.32
36	400000035	: 3.31
37	400000036	: 3.28



The screenshot shows a text editor window with the title bar 'StudentGPA.txt'. The window contains a list of student IDs and their corresponding GPA values. The IDs are listed in the first column, and the GPA values are listed in the second column, separated by a colon. The IDs range from 49964 to 50001, and the GPA values range from 2.98 to 3.58.

Student ID	GPA
49964	3.31
49965	3.23
49966	3.29
49967	3.37
49968	3.21
49969	3.13
49970	3.58
49971	3.41
49972	3.2
49973	3.17
49974	3.26
49975	3.34
49976	3.4
49977	3.28
49978	3.44
49979	3.19
49980	3.19
49981	3.06
49982	3.28
49983	3.22
49984	3.38
49985	3.39
49986	3.37
49987	3.42
49988	3.18
49989	3.01
49990	3.1
49991	3.33
49992	3.46
49993	3.34
49994	3.2
49995	3.52
49996	3.17
49997	3.23
49998	3.25
49999	2.98
50000	3.33
50001	3.28