UNIVERSITÉ Concordia UNIVERSITY

COMP 6521
Advanced Database Technology and Application

Project Report
On
Bitmap Index on LA2
Winter 2020

Professor
Dr. Nematollaah Shiri

Team Members

Student Name	Student Id	Email Id
Raj Mistry	40119206	r_istry@encs.concordia.ca
Piyush Thummar	40125029	p_thumma@encs.concordia.ca
Kunjal Kotadia	40120416	k_kotadi@encs.concordia.ca

Contents:

1. Brief Introduction of the bitmap indexes

- A bitmap index is a special kind of database index that uses bitmaps. Bitmap index for a field F is a collection of bit-vectors of length n, one for each possible value that may appear in the field F. The vector for value v has 1 in position i if the i^{th} record has v in field F, and it has 0 there if not.
- The compensating advantage of bitmap indexes is that they allow us to answer partial-match queries very efficiently. Also, we can answer range queries by performing bitwise AND or OR operation on bit-vectors.
- Unlike most other types of indexes, bitmap indexes include rows that have NULL values. Indexing of nulls can be useful for some types of SQL statements, such as queries with the aggregate function COUNT.

2. Group Member Contribution

➔ Although we have contributed equally throughout whole project but major contribution of each member is as below:
  - **Piyush Thummar (40125029):** The bitmap creation by creating and using partial bitmap indexes on empID, Gender and Department for given files.
  - **Raj Mistry (40119206):** The compression of bitmap indexes for all uncompressed bitmap indexes.
  - **Kunjal Kotadia (40120416):** Duplication removal using bitmap indexes on both the files as well as merging both the files, and at last, sorting of merged file.

3. Implementation of creating and compressing bitmap indexes

3.1 Creating Bitmap Index

- To create a bitmap index, as we know we have to take the attribute for which we want to create the bit vector. We have to take all unique values for that attribute and we will compare each tuple's value for that attribute with these unique values one by one and if the tuple's value matched with that unique value, we will put 1 for that unique value for that record and put 0 if not matched. Here we need to make a bitmap index for 3 attributes: employee Id, gender and department.
- So, in order to create a bitmap for this we have divided the main file containing all the records into chunks and created partial bitmap indexes for it. Then we are merging those partial bitmap indexes in order to get the final bitmap index for the file. To merge those partial bitmap indexes, we take a record from a subfile and compare it with other subfiles that if it's present in another file or not, if its present then we merge that indexes by concatenating them, otherwise we put the zeros for those many records of that partial-index which does not have entry for given record.

3.2 Compression of Bitmap Index

- To compress the bitmap, we have used a common approach called **"run-length encoding".** In which, we represent a run of i 0's followed by a 1, by some suitable binary encoding of the integer i. We concatenate the codes for each run together, and that sequence of bits we get by this, is the encoding of the entire bit-vector.
- To implement this approach, we first determine the length(i) of the first run of 0's followed by 1, then for that how many bits required(j) to represent its binary value. Once we determine this, we add (j-1) 1's followed by one 0 and binary value of i to the output. Then we repeat this step for each run of this bit vector and concatenate it together in order which will give us compressed encoding for that bit vector.
- **E.g.** bitmap for field F is **'0000010001000'.** As we can see here there are 2 runs of i 0's followed by 1. Each of length 5 and 3 respectively. So, for the first run it requires 3 bits to represent a binary of 5 so for that we will add 110 followed by a binary of 5 i.e. 101 so we will get '110101' for the first run. Similarly, for the second run we require 2 bits of bitmap for representing 3 so output for that run is

'1011'. And the final output for bitmap for field F is concatenation of these two outputs which is **"1101011011".**

4. How to use bitmap indexes to merge the records

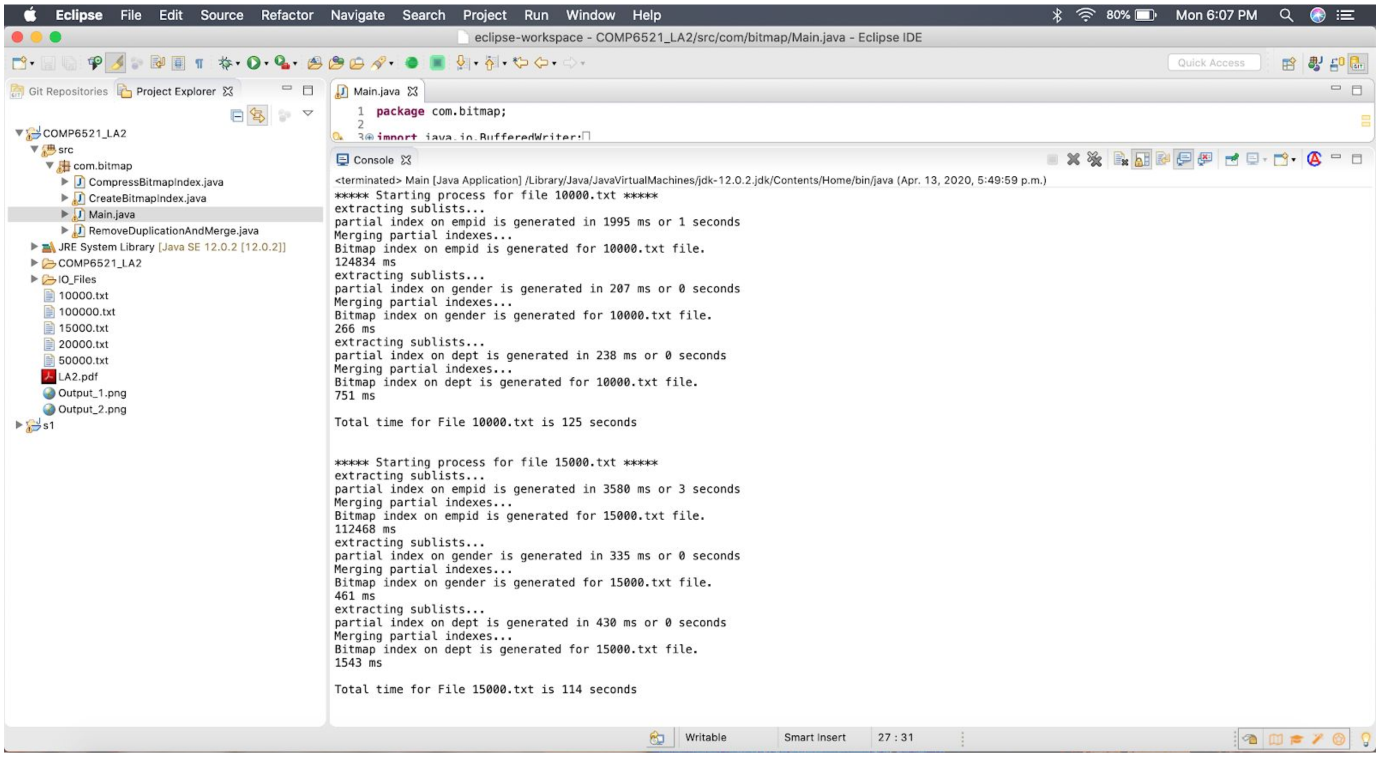
- In order to merge the files, we use the bitmap index of that file as follows: we read each entry from the bitmap file, if the bitmap value is '1', then we take that record and store it in list, and skip if entry is '0'. Now, we take out the latest record from this list and store it in the file. This process must be done for both the input files, which will remove the duplicates from each file and will have the latest copy of record from each duplicate record. Now, we use these new files to merge them. We take each entry from the first file and check that if this entry is present in the second file, if yes, then we choose the latest record from them and write into the new file; and, if not present, then write the current record as it is. Now, we check for each entry of the second file that if the entry for that attribute is already written then skip that record, otherwise write it into the merge file.

5. What method is used for sorting records

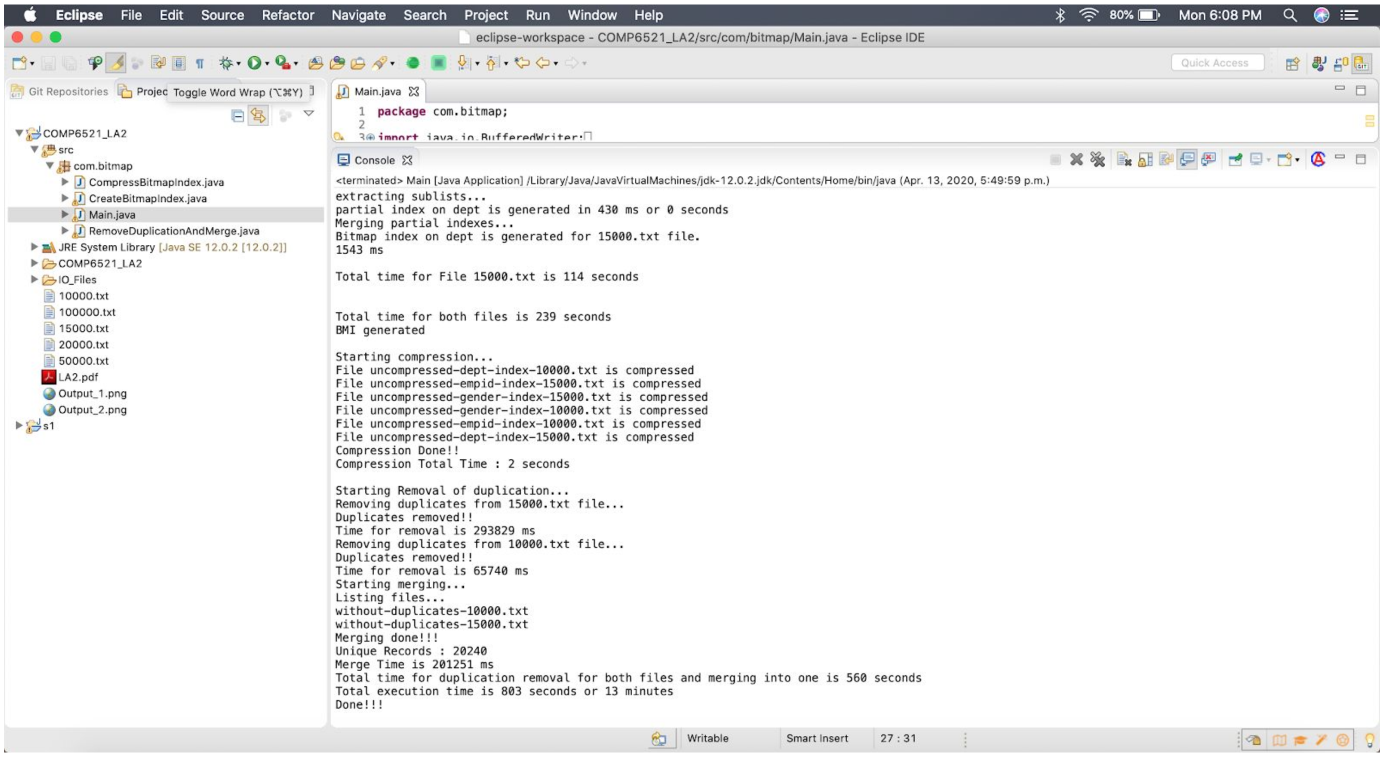
- We have used the in-built java sorting method of Collections class to sort the final records after merging both the files into one.

6. Test Results / Test Cases

- Test-1 : File-1 has 10,000 records and File-2 has 15,000 records.



```
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java (Apr. 13, 2020, 5:49:59 p.m.)
***** Starting process for file 10000.txt *****
extracting sublists...
partial index on empid is generated in 1995 ms or 1 seconds
Merging partial indexes...
Bitmap index on empid is generated for 10000.txt file.
124834 ms
extracting sublists...
partial index on gender is generated in 207 ms or 0 seconds
Merging partial indexes...
Bitmap index on gender is generated for 10000.txt file.
266 ms
extracting sublists...
partial index on dept is generated in 238 ms or 0 seconds
Merging partial indexes...
Bitmap index on dept is generated for 10000.txt file.
751 ms

Total time for File 10000.txt is 125 seconds


***** Starting process for file 15000.txt *****
extracting sublists...
partial index on empid is generated in 3580 ms or 3 seconds
Merging partial indexes...
Bitmap index on empid is generated for 15000.txt file.
112468 ms
extracting sublists...
partial index on gender is generated in 335 ms or 0 seconds
Merging partial indexes...
Bitmap index on gender is generated for 15000.txt file.
461 ms
extracting sublists...
partial index on dept is generated in 430 ms or 0 seconds
Merging partial indexes...
Bitmap index on dept is generated for 15000.txt file.
1543 ms

Total time for File 15000.txt is 114 seconds
```

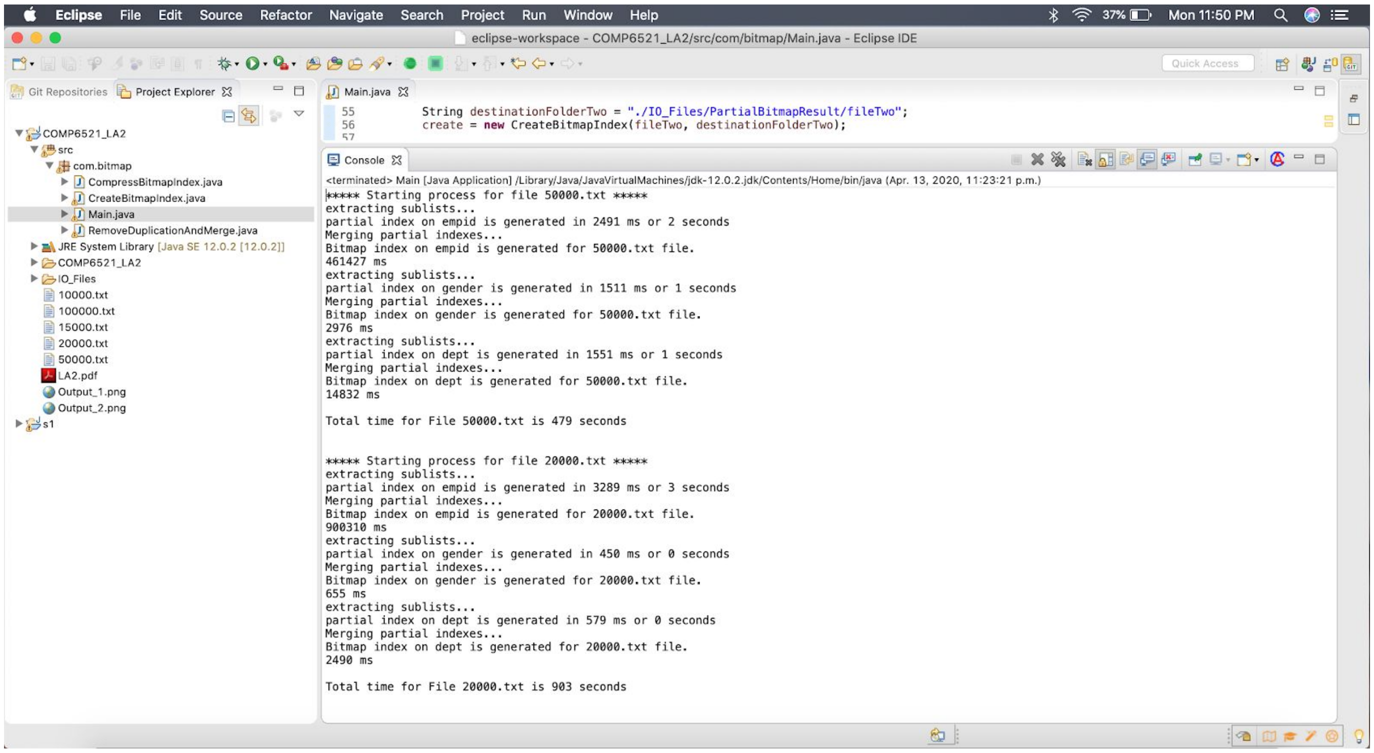


```
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java (Apr. 13, 2020, 5:49:59 p.m.)
extracting sublists...
partial index on dept is generated in 430 ms or 0 seconds
Merging partial indexes...
Bitmap index on dept is generated for 15000.txt file.
1543 ms

Total time for File 15000.txt is 114 seconds


Total time for both files is 239 seconds
BMI generated

Starting compression...
File uncompressed-dept-index-10000.txt is compressed
File uncompressed-empid-index-15000.txt is compressed
File uncompressed-gender-index-15000.txt is compressed
File uncompressed-gender-index-10000.txt is compressed
File uncompressed-empid-index-10000.txt is compressed
File uncompressed-dept-index-15000.txt is compressed
Compression Done!!
Compression Total Time : 2 seconds

Starting Removal of duplication...
Removing duplicates from 15000.txt file...
Duplicates removed!!
Time for removal is 293829 ms
Removing duplicates from 10000.txt file...
Duplicates removed!!
Time for removal is 65740 ms
Starting merging...
Listing files...
without-duplicates-10000.txt
without-duplicates-15000.txt
Merging done!!!
Unique Records : 20240
Merge Time is 201251 ms
Total time for duplication removal for both files and merging into one is 560 seconds
Total execution time is 803 seconds or 13 minutes
Done!!!
```

Total DISK I/Os = 2*{(B(T1)+B(T2)) + (B(T1-without-duplication) + B(T2-without-duplication))}

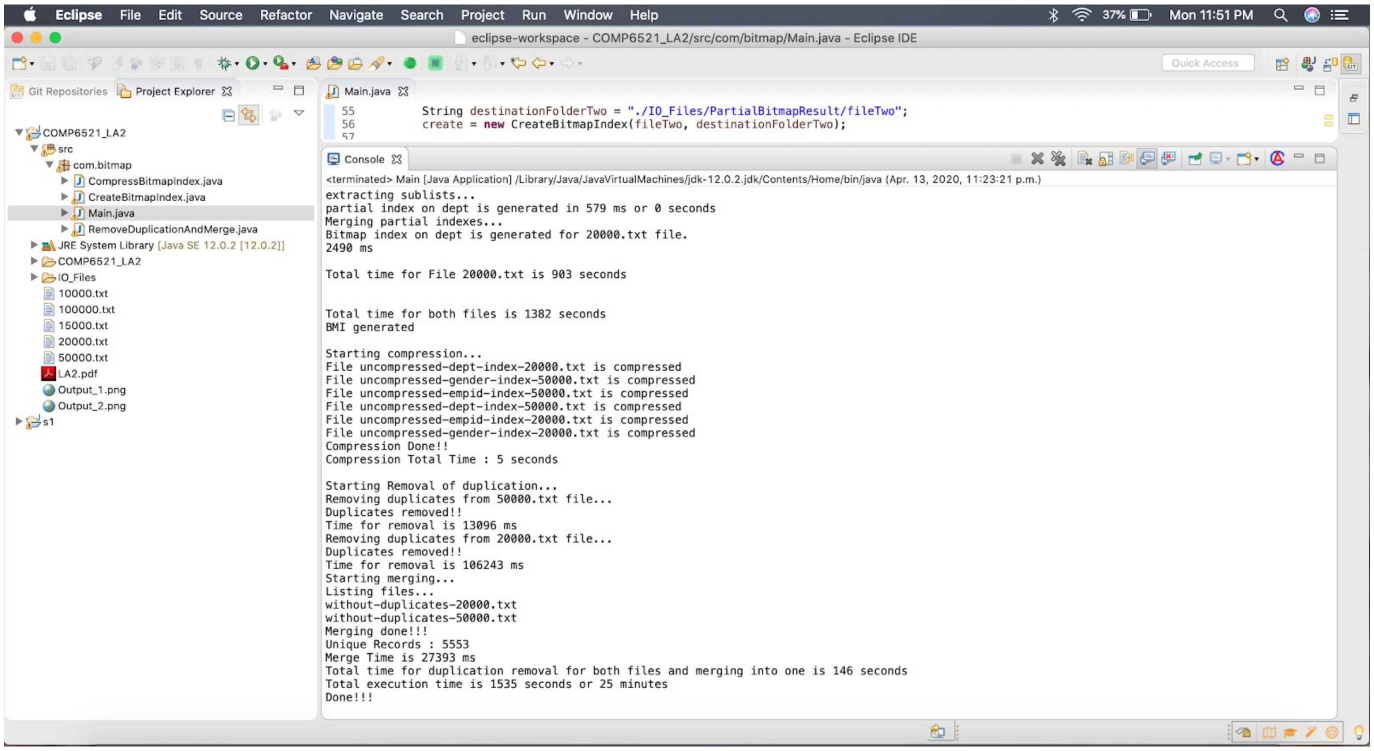
= 2 * {(10000/40 + 15000/40) + (5553/40 + 14886/40)} = 2274 blocks

- Test-2 : File-1 has 50,000 records and File-2 has 20,000 records.



```
String destinationFolderTwo = "./IO_Files/PartialBitmapResult/fileTwo";
create = new CreateBitmapIndex(fileTwo, destinationFolderTwo);
```

```
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java (Apr. 13, 2020, 11:23:21 p.m.)
***** Starting process for file 50000.txt *****
extracting sublists...
partial index on empid is generated in 2491 ms or 2 seconds
Merging partial indexes...
Bitmap index on empid is generated for 50000.txt file.
461427 ms
extracting sublists...
partial index on gender is generated in 1511 ms or 1 seconds
Merging partial indexes...
Bitmap index on gender is generated for 50000.txt file.
2976 ms
extracting sublists...
partial index on dept is generated in 1551 ms or 1 seconds
Merging partial indexes...
Bitmap index on dept is generated for 50000.txt file.
14832 ms

Total time for File 50000.txt is 479 seconds


***** Starting process for file 20000.txt *****
extracting sublists...
partial index on empid is generated in 3289 ms or 3 seconds
Merging partial indexes...
Bitmap index on empid is generated for 20000.txt file.
900310 ms
extracting sublists...
partial index on gender is generated in 450 ms or 0 seconds
Merging partial indexes...
Bitmap index on gender is generated for 20000.txt file.
655 ms
extracting sublists...
partial index on dept is generated in 579 ms or 0 seconds
Merging partial indexes...
Bitmap index on dept is generated for 20000.txt file.
2490 ms

Total time for File 20000.txt is 903 seconds
```



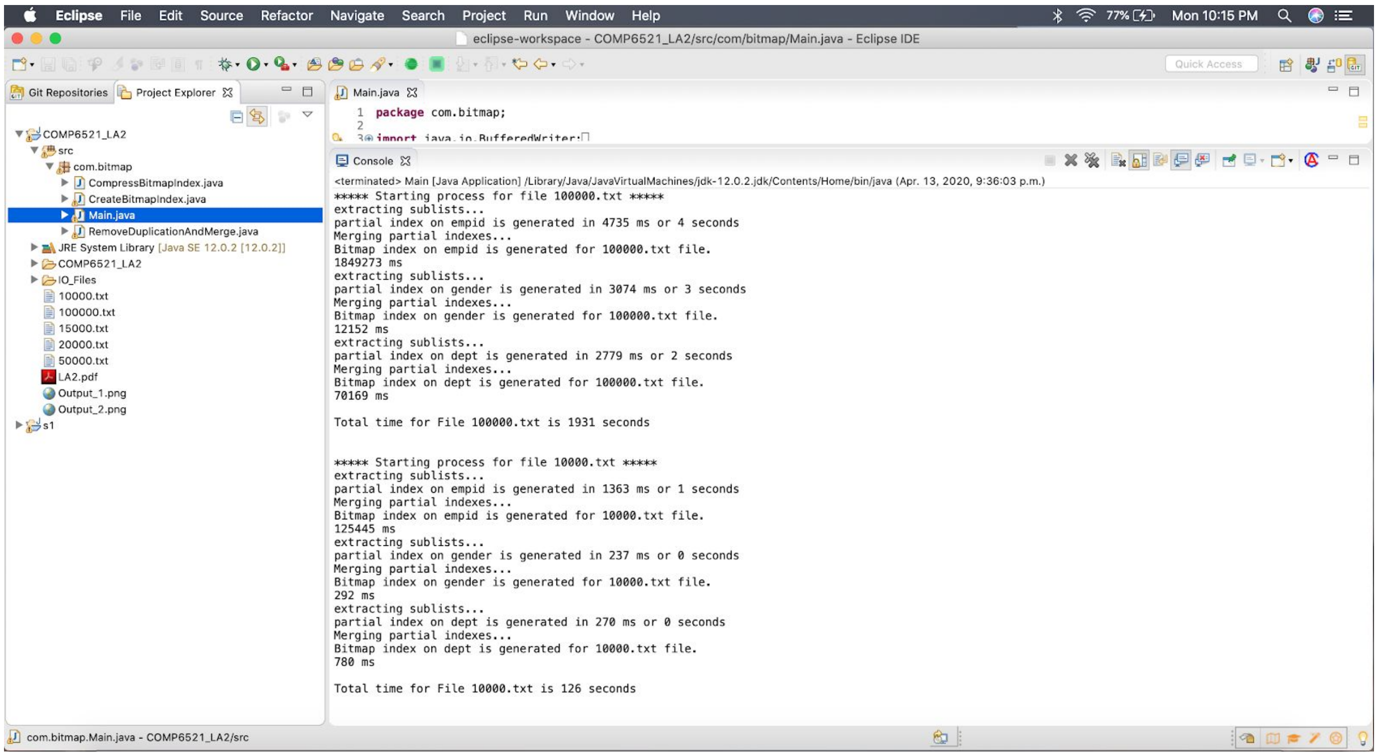
```
String destinationFolderTwo = "./IO_Files/PartialBitmapResult/fileTwo";
create = new CreateBitmapIndex(fileTwo, destinationFolderTwo);
```

```
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java (Apr. 13, 2020, 11:23:21 p.m.)
extracting sublists...
partial index on dept is generated in 579 ms or 0 seconds
Merging partial indexes...
Bitmap index on dept is generated for 20000.txt file.
2490 ms

Total time for File 20000.txt is 903 seconds


Total time for both files is 1382 seconds
BMI generated

Starting compression...
File uncompressed-dept-index-20000.txt is compressed
File uncompressed-gender-index-50000.txt is compressed
File uncompressed-empid-index-50000.txt is compressed
File uncompressed-dept-index-50000.txt is compressed
File uncompressed-empid-index-20000.txt is compressed
File uncompressed-gender-index-20000.txt is compressed
Compression Done!!
Compression Total Time : 5 seconds

Starting Removal of duplication...
Removing duplicates from 50000.txt file...
Duplicates removed!!
Time for removal is 13096 ms
Removing duplicates from 20000.txt file...
Duplicates removed!!
Time for removal is 106243 ms
Starting merging...
Listing files...
without-duplicates-20000.txt
without-duplicates-50000.txt
Merging done!!!
Unique Records : 5553
Merge Time is 27393 ms
Total time for duplication removal for both files and merging into one is 146 seconds
Total execution time is 1535 seconds or 25 minutes
Done!!!
```

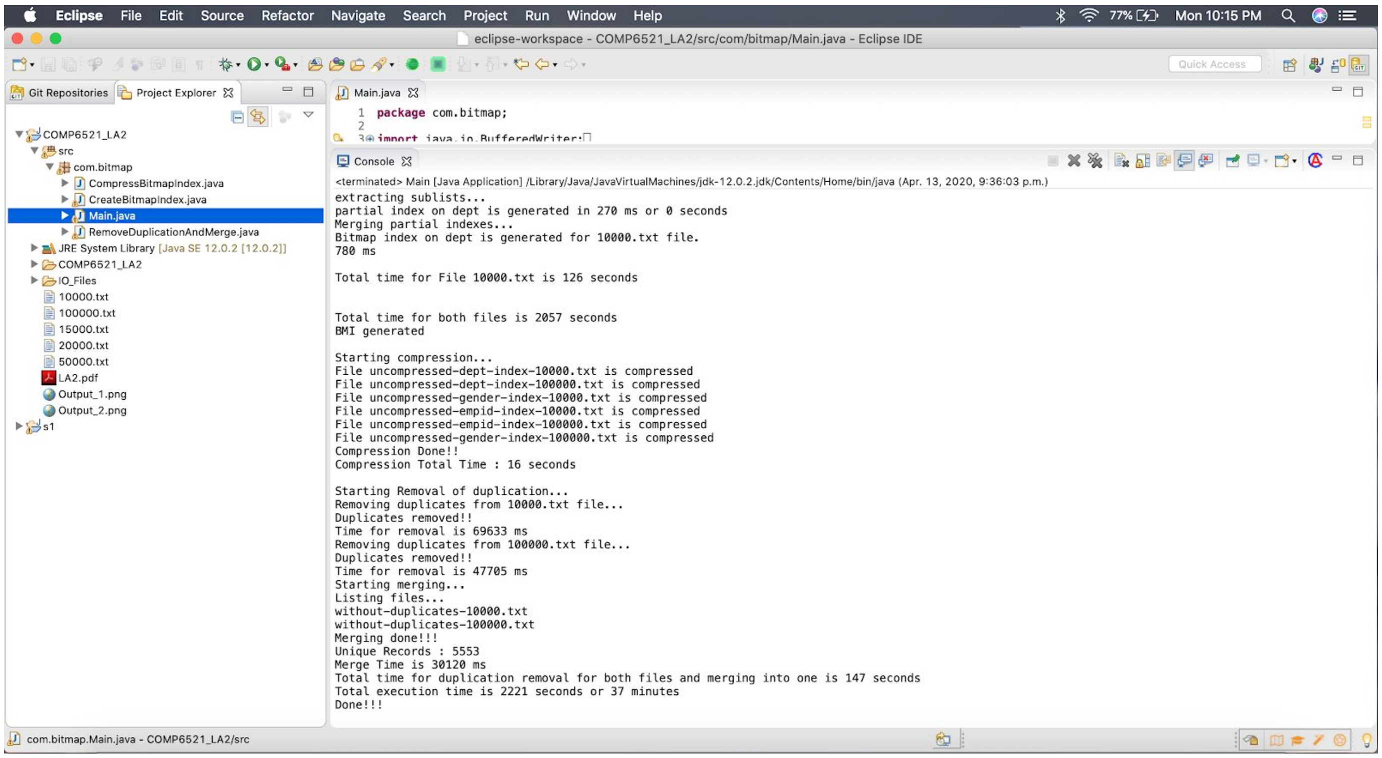
Total DISK I/Os = 2*{(B(T1)+B(T2)) + (B(T1-without-duplication) + B(T2-without-duplication))}

= 2 * {(20000/40 + 50000/40) + (5553/40 + 3517/40)} = 3954 blocks

- Test-3 : File-1 has 100,000 records and File-2 has 10,000 records.



```
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java (Apr. 13, 2020, 9:36:03 p.m.)
***** Starting process for file 100000.txt *****
extracting sublists...
partial index on empid is generated in 4735 ms or 4 seconds
Merging partial indexes...
Bitmap index on empid is generated for 100000.txt file.
1849273 ms
extracting sublists...
partial index on gender is generated in 3074 ms or 3 seconds
Merging partial indexes...
Bitmap index on gender is generated for 100000.txt file.
12152 ms
extracting sublists...
partial index on dept is generated in 2779 ms or 2 seconds
Merging partial indexes...
Bitmap index on dept is generated for 100000.txt file.
70169 ms

Total time for File 100000.txt is 1931 seconds


***** Starting process for file 10000.txt *****
extracting sublists...
partial index on empid is generated in 1363 ms or 1 seconds
Merging partial indexes...
Bitmap index on empid is generated for 10000.txt file.
125445 ms
extracting sublists...
partial index on gender is generated in 237 ms or 0 seconds
Merging partial indexes...
Bitmap index on gender is generated for 10000.txt file.
292 ms
extracting sublists...
partial index on dept is generated in 270 ms or 0 seconds
Merging partial indexes...
Bitmap index on dept is generated for 10000.txt file.
780 ms

Total time for File 10000.txt is 126 seconds
```



```
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java (Apr. 13, 2020, 9:36:03 p.m.)
extracting sublists...
partial index on dept is generated in 270 ms or 0 seconds
Merging partial indexes...
Bitmap index on dept is generated for 10000.txt file.
780 ms

Total time for File 10000.txt is 126 seconds


Total time for both files is 2057 seconds
BMI generated

Starting compression...
File uncompressed-dept-index-10000.txt is compressed
File uncompressed-dept-index-100000.txt is compressed
File uncompressed-gender-index-10000.txt is compressed
File uncompressed-empid-index-10000.txt is compressed
File uncompressed-empid-index-100000.txt is compressed
File uncompressed-gender-index-100000.txt is compressed
Compression Done!!
Compression Total Time : 16 seconds

Starting Removal of duplication...
Removing duplicates from 10000.txt file...
Duplicates removed!!
Time for removal is 69633 ms
Removing duplicates from 100000.txt file...
Duplicates removed!!
Time for removal is 47705 ms
Starting merging...
Listing files...
without-duplicates-10000.txt
without-duplicates-100000.txt
Merging done!!!
Unique Records : 5553
Merge Time is 30120 ms
Total time for duplication removal for both files and merging into one is 147 seconds
Total execution time is 2221 seconds or 37 minutes
Done!!!
```

Total DISK I/Os = 2*{(B(T1)+B(T2)) + (B(T1-without-duplication) + B(T2-without-duplication))}

= 2 * {(100000/40 + 10000/40) + (5553/40 + 5553/40)} = 6506 blocks

7. Performance Evaluation

- As from LA-1 and LA-2, we came to the conclusion that if there are a small number of records, then LA2 is very much useful and faster. On the other side, when we have a large number of records, then better to go with LA-1.
- Comparing about Disk I/Os, it requires 3*(B(T1) + B(T2)) for LA1, whereas, for LA2, it requires 2*{(B(T1)+B(T2)) + (B(T1-without-duplication) + B(T2-without-duplication))} Disk I/Os.
- Comparing about Performance Time, in LA-1, for 10,000 and 15,000 records which has combinely around 20,148 records take 10-15 seconds, whereas, in LA-2, it takes 800-820 seconds; and, for 500,000 and 1,000,000 tuples with combinely 841,455 unique records only takes approximately 130 seconds, while in LA-2, it takes about 6 to 7 hours.
- Thus, to wrap up the discussion, we can say that it is good to go with the LA-2 (Bitmap) approach for small numbers of records, otherwise, LA-1 (TPMMS) approach is the best for large files.