Group 31

Project Phase – 3 Report

March 27

2016

Team Members:

1. Shreyas Talele 1209355546
2. Mihir Bhatt 1209422146
3. Varun Gaur 1210414176
4. Shruti Mahajan 1210431622
5. Anique Tahir 1209442595

ABSTRACT

MINIBASE is a software build on java (1.8) for implementing a relational database management system which help us to analyze how DBMS works. It inherently has two join algorithms namely sort-merge join and nested-loop join. Primitively it implements nested-loop join which bears high computational costs for queries dealing in inequality. Hence to optimize results we have implemented a join algorithm inequality (IE) Join on minibase. Implementation comprises two relations with inequality joins on one and two predicates. A special case of IE join i.e. self-join has also been illustrated and implemented in this phase. The report also highlights the difference in execution (time and memory) of single and double predicate query with different data-sets using nested-loops and IE Joins Algorithm.

1. INTRODUCTION

Joins are a fundamental database operation which are frequently used in database queries. Join methods implemented in Minibase are sort-merge and nested-loop algorithm. As a novel approach to joins; inequality joins (IE) has been illustrated and implemented in this phase III of the project. Different Join algorithm has been explained below.

1.1 Sort-Merge Join Algorithm

The relations to be joined are first sorted on the join attribute and later the tuples which satisfy the join condition are merged together in the join tuple.

1.2 Nested-Loop Algorithm

This algorithm hosts the join in the form of iterative structures. Loops are in a nested form each iterates through a relation contained within it. The join condition is placed in the innermost loop where the tuples satisfying the inequality are joined together.

1.3 IE Joins

IE join algorithm is a fast and space efficient algorithm. The attributes to be joined are sorted and stored prior the join operation.

- Sort tuples according to the operator given in the query and store it in L1, L2.
- Initialize bit array and permutation array which stores position of L2 w.r.t its position in L1.
- Scan the tuples in the permutation array sequentially and mark true the corresponding bit in the bit array.
- Scan the bit array towards the right of the present bit and join the tuple in consideration with the tuples, respective bits of which are set in the array.

1.4 IE-Self Join

This is a special case of Inequality join which is used in the queries which needs to create its own sub-query. Computing a sub-query is instead replaced by a join operation which joins a table to itself and as and when it comes across the qualifying tuples; they're appended to the output set. For example the given query Q1 can be replaced by a query Q2[1].

Q1:

```
SELECT employee_name FROM employee
 WHERE employee_location in( SELECT employee_location
FROM employee WHERE employee_name = "Joe")
```

Q2:

```
SELECT e1.employee_name
FROM employee e1, employee e2
WHERE e1.employee_location = e2.employee_location
AND e2.employee_name="Joe";
```

The query Q2 is a more simplified version of Q1 which joins the table employee to itself in order to retrieve the results and thus gives computation benefits.

2. IMPLEMENTATION DETAILS

2.1 Query Loading and Query Parsing

Query file is parsed from given input query.txt file from home directory and data is loaded from corresponding relations mentioned in query and data is then stored in corresponding heap files. Given predicates are parsed and used for query evaluation.

2.2 Join Implementation

A. Task 1a

Query 1a has two relations and single predicate. Condition expression is prepared from given predicate. Projection attributes are selected according to query.

Operations performed consecutively: FileScan, Projection, and Nested Loop Join. It first performs file scan on first relation's heap file.

Using file-scan iterator and second relation's heap file, it performs nested loop join. Using its iterator, it outputs its all tuples satisfying inequality condition.

Data Structures:

- Heap Files (relation's data)
- Tuples(satisfying inequality join).

Assumptions:
- It supports exact two relations.
- It supports exact two projections.
- It supports columns with data types integers only.
- It supports <,>,<= and >= inequality operators in given predicate.

Limitations:
- It does not support more than two relation.
- It does not support more than two projections.
- Columns with data types other than integers are not supported.

Optimization:
- It can be implemented using sort-merge join which will reduce time complexity.

Known issues:
- Not able to optimize this query using Sort-Merge Join.

Extra Operation:
- It handles duplicates in given relation .txt file.

B. Task 1b

Query 1b has two relations and two predicates. Condition expression is prepared from given predicates. Projection attributes are selected according to query.

Operations performed consecutively:: FileScan, Projection, and Nested Loop Join
It first performs file scan on first relation's heap file.

Using filescan iterator and second relation's heap file, it performs nested loop join. Using its iterator, it outputs its all tuples satisfying inequality condition.

Data Structures:
- Heap Files (relation's data)
- Tuples (satisfying inequality join).

Assumptions:
- It supports exact two relations.
- It supports exact two projections.

- It supports columns with data types integers only.
- It supports <,>,<= and >= inequality operators in given predicate.

Limitations:

- It does not support more than two relations.
- It does not support more than two projections.
- Columns with data types other than integers are not supported

Known issues:

- Not able to optimize this query using Sort-Merge Join.

Optimization:

- It can be implemented using sort-merge join which will reduce time complexity.

Extra Operation:

- It handles duplicates in given relation .txt file.
- It supports both AND/OR operators in given predicate

C. Task 2a & 2b

Task 2a and 2b have been clubbed together and the desired algorithm toggles on the basis of an if-else statement which distinguishes on the basis of the predicate length whether it is a single or double predicate. The main function is IESelfJoin() in the main class IEJoinDriver which takes in the the data file and parsed query as an input and produces results on the console

- **Single predicate Self Join**

Design Choices

The tuples are read and sorted in an ascending order into a heap file named sortedHeapFile. Eventually the columns on which the predicate has to be applied is extracted from the heap file; compared with the preceding tuples (op in >/≥) or the following tuples (op in </≤) and appended in the output.

Data Structures:

- Heap Files (relation's data)
- Tuples (satisfying inequality join).

- **Double predicate Self Join**

Design Choices

The two different sort orders required are stored The tuples contained in the predicate are maintained in two sorting orders as per the column attributes required by the given predicate. Initially we sort the file with respect to the second column attribute and then we fetch the order of occurrence of each tuple as in the order of first predicate column which is stored in the bit array. As we traverse the second predicate column, we set the corresponding bit which is nothing but that record's position in L1. For each and every tuple position in the bit array we traverse towards the right and join it with the ones whose respective bits are set.

Data Structures:

- Heap Files (relation's data)
- Tuples (satisfying inequality join).

C.3 Optimzation

In our implementation the permutation array has been calculated with a linear complexity O(n) which is an optimization on the computation complexity O(nlogn) presented in the research paper given.

D. Task 2c.

Query 2c has two relations and two predicates. Condition expression is prepared from the given predicates and operators. Projection Attributes are selected according to the query.

Implementation

- File Scan: The algorithm starts with the file scan phase wherein, it reads the files given into the query into a heap file, called reserves and sailors.
- Sort phase: It reads the heap files created earlier and then the sort is performed on the first columns of both the predicates and the data is stored back to sorted file (L1 and L1'). Depending upon the operator in the query, the sort is either in ascending order or descending order.
- It follows the same operation on the second columns of both the predicates and stores the sorted data in L2 and L2' heap-files, respectively. Again the sorting (ascending/descending) is dependent on the operator provided in the query.
- Permutation Array: While creating L2 and L2', it also calculates and stores Permutation Matrix (P and P'). Permutation Array P, is the relative position of L2 record in L1. Similarly, P' is the relative of L2' record in L1'.
- Offset Calculation: Using L1, L1', L2 and L2', the code calculates the relative of L1 in L1' (O1) and L2 in L2' (O2)
- Bit Array Initialization

IEJoin Algorithm :

The permutation arrays (P & P') and Offset Array (O1 and O2) are given as input to the IEJoin algorithm, which sets the bits in the Bit Array (B) and brings up the join result depending the condition expression and operators.

Operator Combination: (1,4); (4,1); (2,3); (3,2); (1,1); (2,2); (3,3); (4,4)

Data Structures:

- Heap files – For storing Relation's data and Sorted files.
- Arrays – For Permutation Matrix (P, P') and Offset Matrix (O1, O2) and Bit Array (B)
- Map <Key, Index> - For storing the mapping of record ids and index in L1/L1'
- List<RID> -For storing and retrieving RIDs in order of L1'.

Assumptions:

- Exactly 2 relations are provided.
- All data types are Integers only
- Exactly 2 Columns predicates are provided
- Exactly 2 Operators are provided
- Exactly 2 Output projections are specified

Limitations:

- Supports exactly 2 relations
- Supports exactly 4 columns in each relation.
- Doesn't support any other data type attributes other than Integer
- Doesn't support for operator combinations from any combination formed from set (1, 4) and set (2,3). For example: 1 and 2 ; 3 and 4; 1 and 3 etc.

Optimization:

- Permutation Matrix - Complexity Reduced from O(n^2) to O(n), by storing the Key(rid) and index of the record in Map, while evaluating L1 and L1'.
- Offset Matrix – Complexity Reduced from O(n^2) to O(n log n) using Divide and Conquer approach
- Retrieving Records for 2nd Relation – Complexity reduced from O(n^2) to O(n), by storing the Record IDs in L1' order in a List.

E. Task 2c.

Task 2d was about optimizing the actual IEJoin algorithm to run efficiently. As, most of the required optimizations were considered while implementing IEJoin, we have only implemented Bloom Filter, presented in the paper on top of our Optimized IE Join.

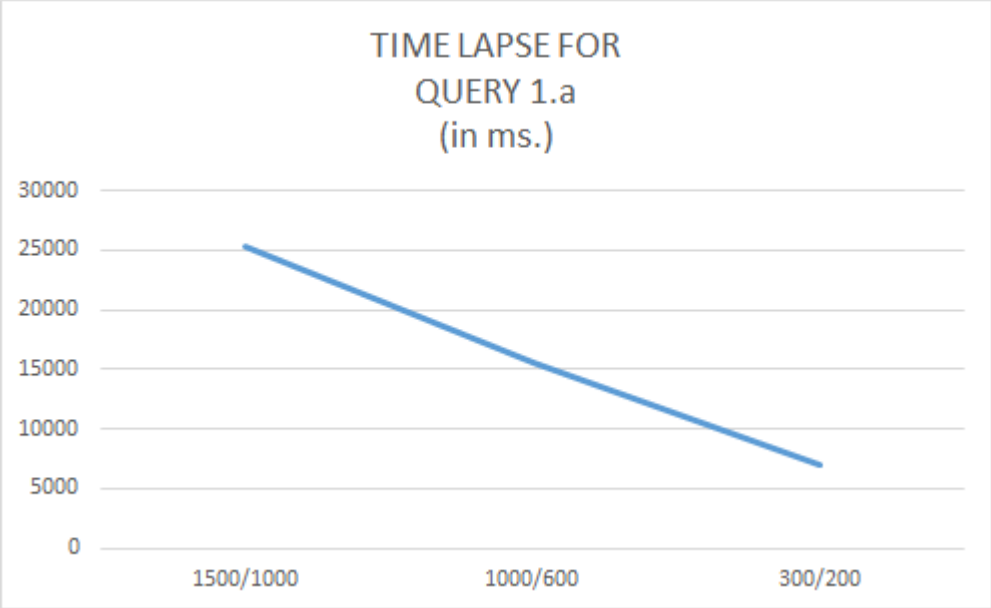
Bloom Filter Implementation (Optimization)

In normal IEJoin, we scan each bits of Bit Array on the right of the current position, while considering for the output. This can cause unnecessary cost in computing and scanning the subarray runs of bit 0. To avoid this, we combine a chunk of Bits into a Single Bit and scan the Bloom Filter Array. Only when the bit is 1 in bloom filter matrix, the algorithm dives in to scan the 1 bits in order. This can help in reducing some computation cost for Bit Arrays having larger length of 0 bits.

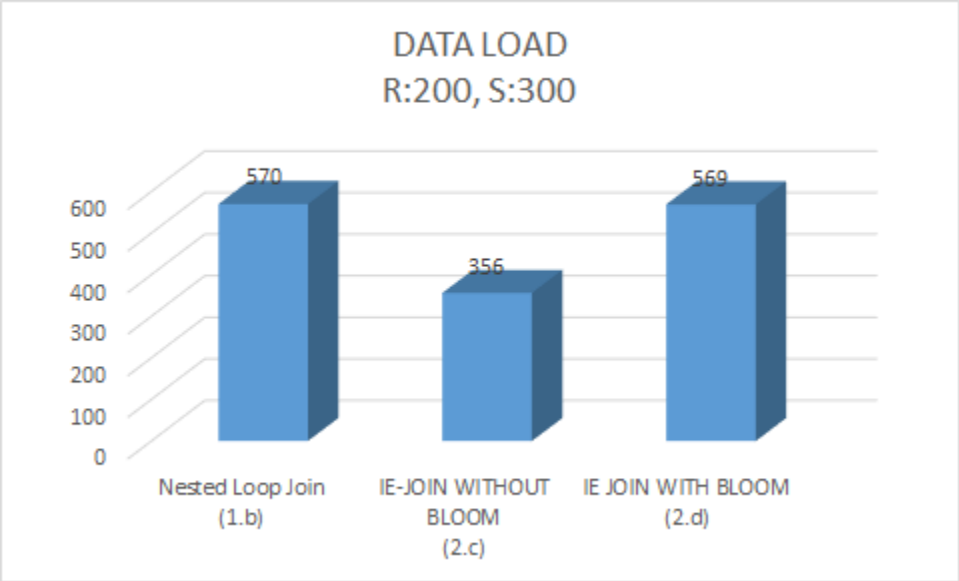
3. EXPERIMENTS

3.1 Individual Tests

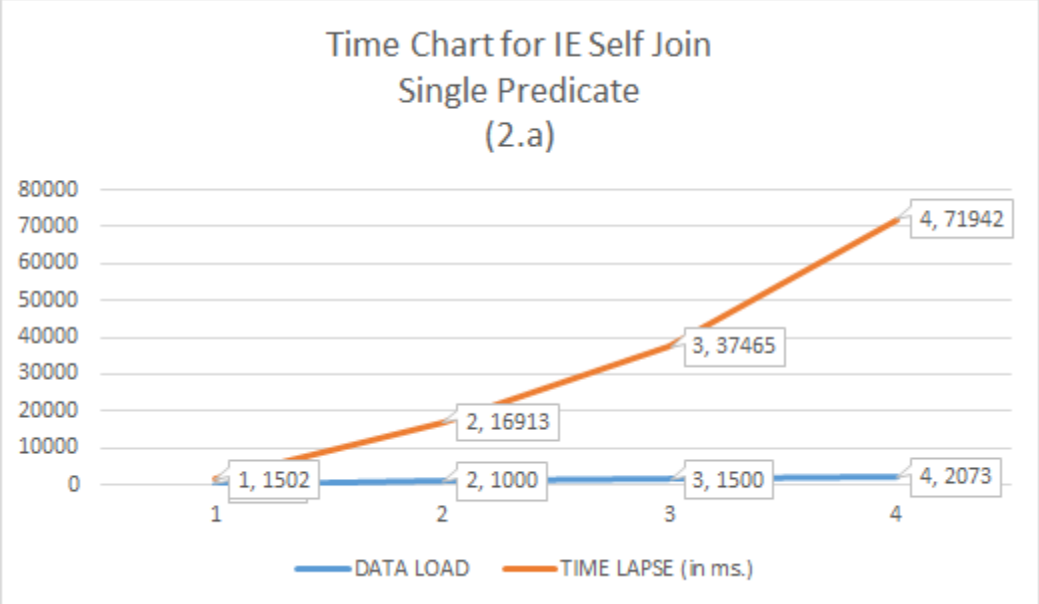
- Task 1a – The following figure shows the implementation of Nested-Loop Join with Single Predicate. The graph indicates the plotting of Number of Records in the relations against the Execution Time.



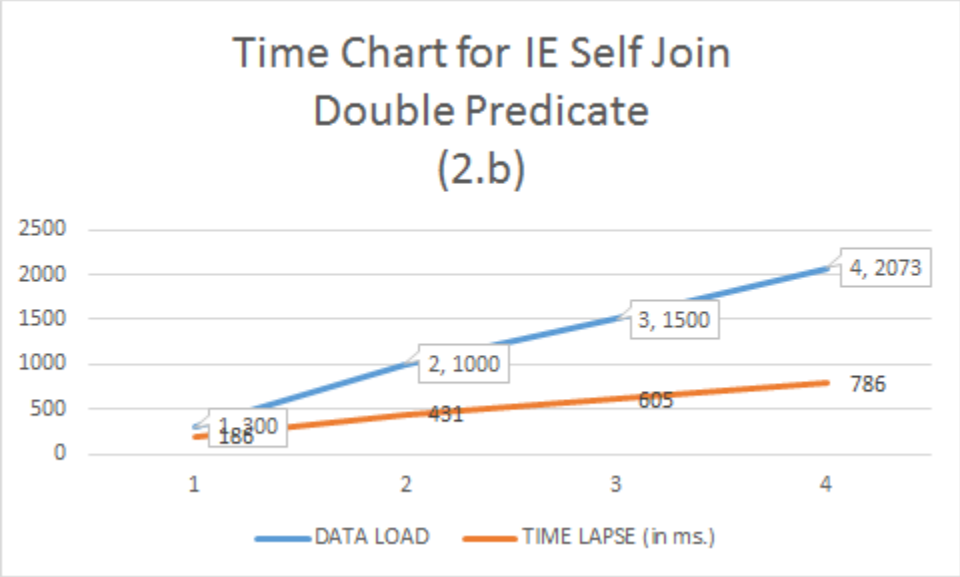
- Task 1b – The following figure shows the implementation of Nested-Loop Join with Double Predicate. The graph indicates the plotting of Varying Number of Records in the relations against the Execution Time.



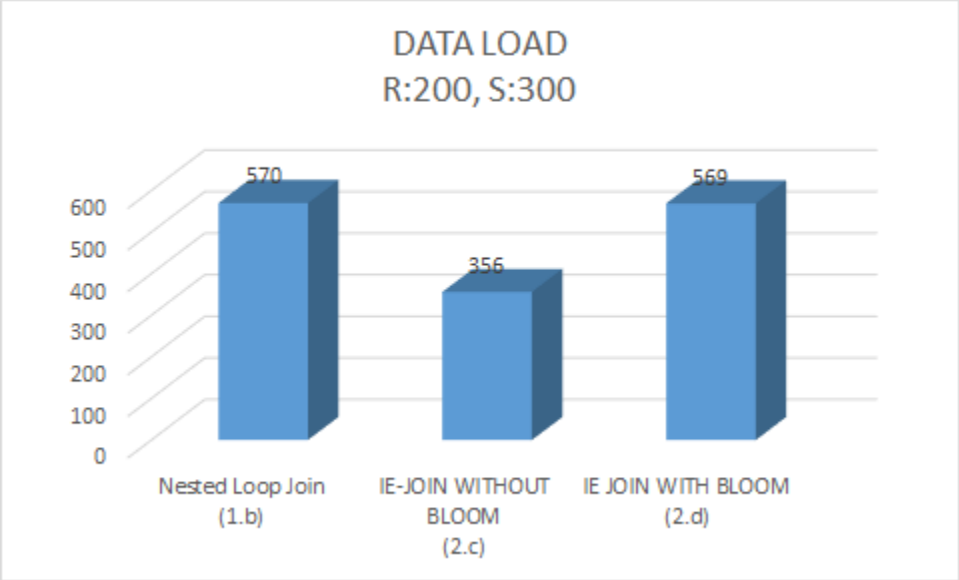
- Task 2a – The following figure shows the implementation of IESelfJoin with Single Predicates. The graph indicates the plotting of Varying Number of Records in the relations against the Execution Time.



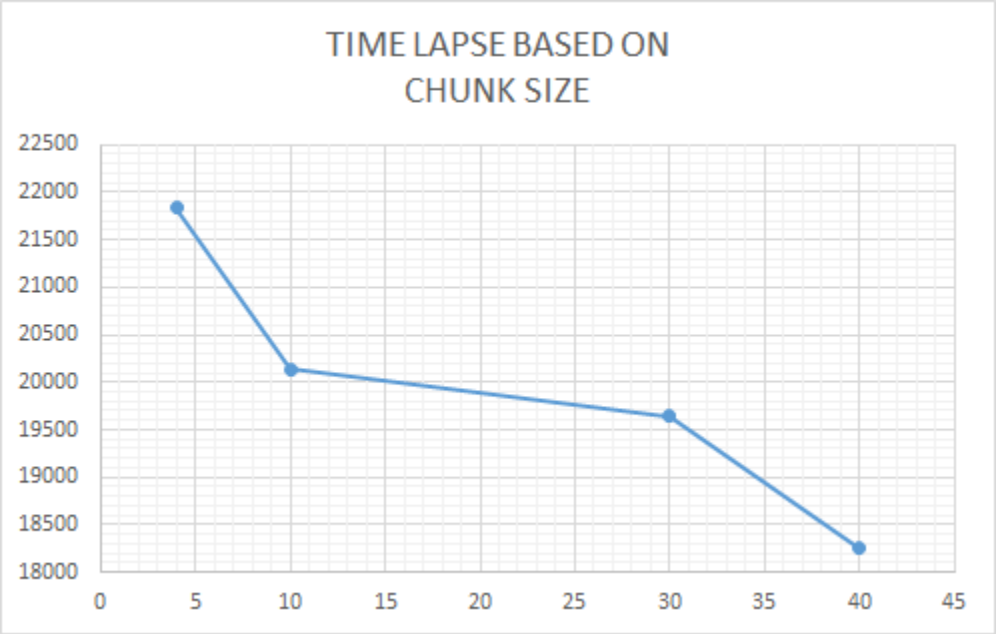
- Task 2b – The following figure shows the implementation of IESelfJoin with Double Predicates. The graph indicates the plotting of Number of Records in the relations against the Execution Time.



- Task 2c – The following figure shows the implementation of IEJoin with Double Predicates. The graph indicates the plotting of Number of Records in the relations against the Execution Time.



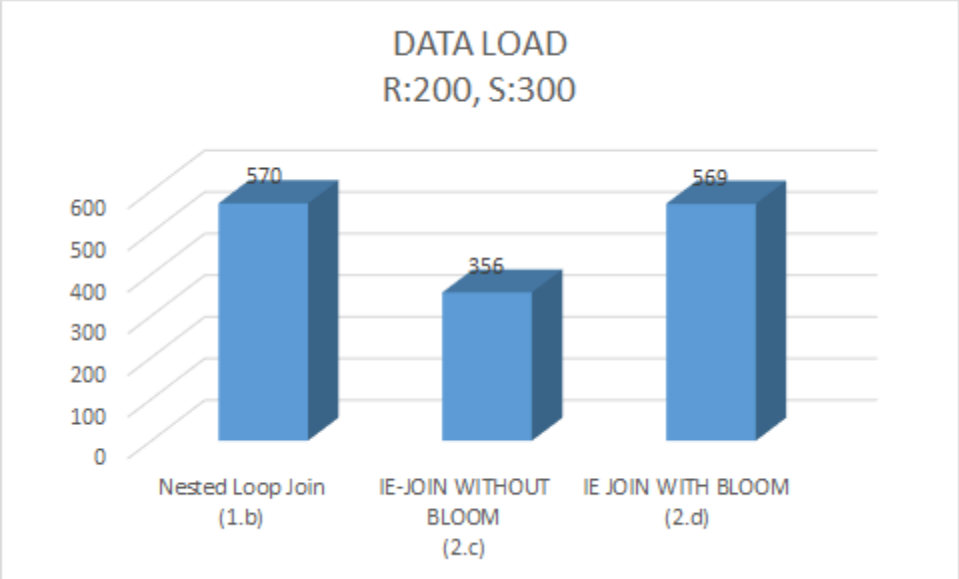
- Task 2d – The following figure shows the implementation of Bloom Filter over IEJoin done in Task 2c. From the graph, it is clear that with increasing Chunk Size for Bloom Array Filter, the execution time decreases.

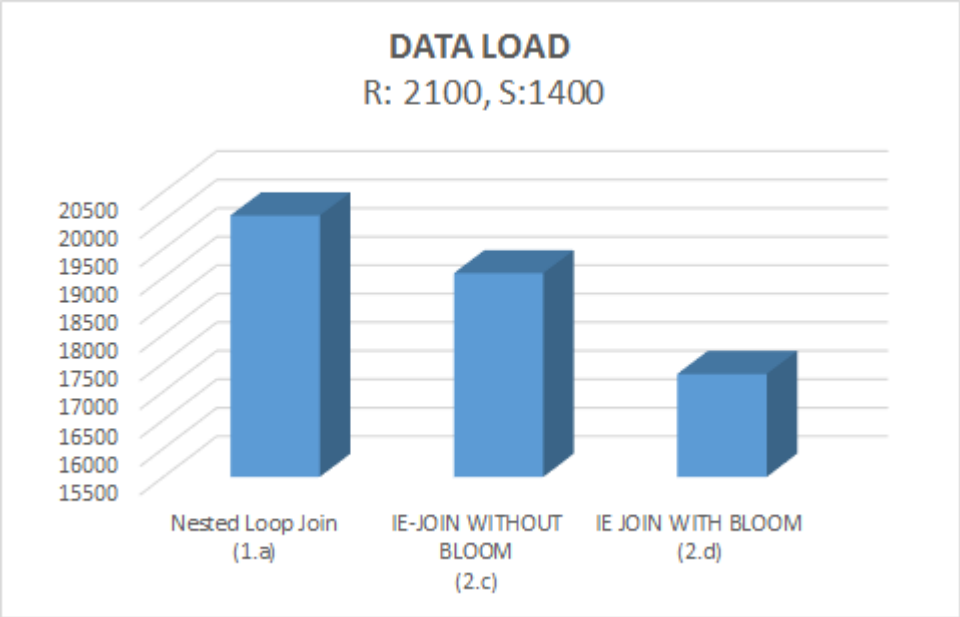


3.1 Comparison – Nested-Loop Join vs IE Join vs IE Join Bloom Filter (Double Predicates)

The following two figures show the implementation of IEJoin and Nested-Loop Join with Double Predicates (Task 1a, 2c and 2d). The graph indicates the plotting of task 1a, 2c and 2d for the relation data load of (2100 & 1400) and (200 & 300) against the execution time.

The test is passed successfully with Nested Loop Join having maximum execution time, following 2c and 2d. IE Join execution time is better than Nested Loop Join. Finally, Bloom Filter is much better than both.





4. CONCLUSION

We have successfully implemented all the given tasks and demonstrated the internal implementation of a Join Query on a single and double predicates, performed by the database. We performed Nested-Loop Join and Inequality Join. All the tests completed successfully for the both.

We conclude that the execution time of the join query is more in nested-loop join than in Inequality join, having same number of predicate columns and similar operations.

5. REFERENCES

1. http://www.tutorialspoint.com/sql/sql-self-joins.htm
2. Database System Concepts by Abraham Silberschatz, Henry Korth, S. Sudarshan, Tata McGrawHill.
3. Database Management Systems by Raghu Ramakrishnan and Johannes Gehrke
4. http://pages.cs.wisc.edu/~dbbook/openAccess/Minibase/bufMgr/bufMgr.html
5. http://www.vldb.org/pvldb/vol8/p2074-khayyat.pdf
6. Internet