Volume 3, No. 6, June 2012

Journal of Global Research in Computer Science

ISSN-2229-371X

RESEARCH PAPER

Available Online at www.jgrcs.info

ADAPTIVE JOIN OPERATORS FOR RESULT RATE MAXIMIZATION

Ms. Pallavi D.Umap*[1], Prof.Dr.G.R.Bamnote[2]

*[1]Information Technology, Prof .Ram Meghe Institute of Technology & Research, Badnera(Amravati),Maharashtra, India.
p.p.takalkhede@gmail.com[1]

[2]Computer Sci & Engg, Prof .Ram Meghe Institute of Technology & Research, Badnera(Amravati),Maharashtra, India.
grbamnote@rediffmail.com[2]

Abstract: - This work is focused on how join operator works in a single, homogeneous and heterogeneous environment. Adaptive join algorithms have recently attracted a lot of attention in emerging applications where data is provided by autonomous data sources through heterogeneous network environments. In traditional join techniques, they can start producing join results as soon as the first input tuples are available, thus improving pipelining by smoothing join result production and by masking source or network delays. In this work, Evaluation of the performance and comparison of Multiway join (MJoin), Double Index Nested Loop Reactive Join (DINER), and Multiple Index Nested Loop Reactive Join (MINER). DINER combines two key elements: an intuitive flushing policy that aims to increase the productivity of in-memory tuples in producing results, and a novel re-entrant join technique that allows the algorithm to rapidly switch between processing in-memory and disk-resident tuples, thus better exploiting temporary delays when new data is not available. MINER outperforms in comparison with the previous join algorithms in producing result tuples at a significantly higher rate, while making better use of the available memory.

Keywords: - Query processing, Streams, Joins

INTRODUCTION

In Real systems, it is difficult to maintain all the data is stored in one large table. To do so would require maintaining several duplicate copies of the same values and could threaten the integrity of the data .Instead, IT department everywhere almost always divide their data among several different tables. Because of this, a method is needed to simultaneously access two or more tables by using join operation. Join is a means for combining fields from two tables by using values common to each. Join operation is considered as one of the fundamental operations of relational databases and it is also difficult operation to efficiently implement. Joins are one of the basic constructions of SQL and databases such as, they combine records from two or more database tables into one row source, one set of rows with the same columns and these columns can originate from either of the joined tables as well as be formed using an expressions and built-in or user-defined functions.

Joins are used for joining records or fields from two or more tables in a database by using a value common to both the tables and the result set can be stored or saved in a table [1].

There are four types of joins and they are specified by ANSI (American National Standard Institute) and they are INNER, OUTER, LEFT, and RIGHT. Inner join are further classified into equi join, natural join and cross join. Outer join are further classified as left outer join, right outer join and full outer join. Two tables are used as an example of joins; they are Dept ID column of the Emp table and Dept table.

Emp Table		**Dept Table**	
LastName	**DeptId**	**Dept Id**	**DeptName**
Aa	11	11	Sales
Bb	13	13	Engineering
Cc	13	14	Clerical
Dd	14	15	Marketing
Ee	14		

Figure. 1 Example of Join

Inner join are considered as a common operation of join and they are also a default type of join based on the predicate. They combine the values of two tables and the results are kept in new table. Inner join has both explicit join notation and implicit join notation.

Outer join does not expect any matching record and they does not require each record in two tables to be joined to have a matching record. Outer join does not have Implicit join notation. Explicit join notation and implicit join notation are the ways of expressing join syntax and they are specified by SQL explicit join notation uses the keyword “JOIN” and “On” [1]

Select * from Emp INNER JOIN Dept On Emp.DeptID = Dept.DeptID;

Implicit join notation list the join table and they use select statement:-
Select * from Emp, Dept Where Emp.DeptID = Dept.DeptID;

1. Adaptive Join**:** Adaptation schemes for join queries are significantly more complicated to design and analyze compared to those for selection ordering for several reasons.

The key performance of adaptive joins is rapid availability of first results and a continuous rate of tuple production. It overcomes the situation like initial delay, slow data delivery or bursty arrival, which can affect the efficiency of join [2]

It is used for fast data delivery from one location to another location.When first input tuple is available then starts its joining process compared to traditional joining processes.

Motivation:

Some additional challenges in adaptive joins compared to traditional joins [3] are: The input relations are provided by autonomous network sources. The implication is that one has little or no control over the order or rate of arrival of tuples. Data is transported through unreliable network environment. It is often unsuitable or in-efficient because most traditional join algorithms cannot produce results until at least one of the relations is completely available, the complete data might be available after a long time. Sometimes these algorithms are unusable, if data is completely available but they produce partial results. The availability of partial join results is important for wide range of applications.

Their main advantage over traditional join techniques is that, they can start producing join results as soon as the first input tuples are available, thus improving pipelining by smoothing join result production and by masking source or network delay.

Objectives:

a) Experimental study of DINER(Double Index Nested Loop Reactive Join)
b) Calculate the performance comparison of the DINER, MINER and MJoin.
c) Evaluate the performance of MINER (Multiple Index Nested Loop Reactive Join)

Dissertation work:

The aim of implementing and optimizing of MINER algorithm and comparing it with the DINER and MJoin. In MINER, It uses homogeneous database for higher quality join results. All the data is stored into the buffer and data is fetched according to their index number and apply a join query.

In DINER (Double Index Nested loop Reactive Join), it uses heterogeneous database and it is a novel adaptive join algorithm that supports both equality and range join predicates.The feature of this DINER algorithm is that they are unblocking and they deal with adaptive. They can produce join result, if the one relation is completely arriving. [4]

In MJoin, It uses single database for joining two or more tables. Firstly, consider two tables for joining and this result is stored in the third table. Then, this third table is join with the another table i.e. fourth table.

LITERATURE REVIEW

Existing Join Techniques:

The main three categories of join algorithms are

a) Nested-loop join algorithm
b) Sort-merge join algorithm
c) Hash-based join algorithm

Nested-Loop Join Algorithm [1]:- Nested-loop join is considered as a one of the simplest algorithm of join where, for each record of the first table the entire records of the second table has to be scanned. This process is repeated for each and every record of the first table that is for all the first table records. The loop is of two levels and they are outer loop and the inner loop. First table loop is called as outer loop and the second table loop are called as inner loop. As this, Nested loop join algorithm has a repeated input/output scans of one of the table. They are considered as inefficient.

Let the two tables be A and B, then the algorithm of Nested-loop algorithm are as for each record of table A

```
Read record from table A
        For each record of table B
        Read record from table B
        Compare the join attributes
        If matched
        Then
        Store the records
```

Example: - Consider schema of two tables “Customers” and “Sales”

```
Create Table Customers (Cust_Id int, Cust_Name varchar (10))
Insert Customers values ( 1,’John’)
Insert Customers values (2,’Henry’)
Insert Customers values (3,’Tom’)
```

Another table is ‘Sales’,

```
Create Table Sales (Cust_Id int, Item varchar (10))
Insert Sales values ( 2,’Camera’)
Insert Sales values (3,’Computer’)
Insert Sales values (3,’Monitor’)
Insert Sales values (4,’Printer’)
```

Query is written as:-

```
Select * from Sales S inner join Customers C on S.Cust_Id = C.Cust_Id
```

In above example, the outer table is “Customers” while the inner table is “Sales”. Thus, it begins by scanning the “Customers” table. It takes one customer at a time and, for each customer, it scans the “Sales” table. Since, there are 3customers; it executes the scan of the “Sales” table 3 times. Each scan of the “Sales” table returns 4 rows. It compares each “Sales” to the current “Customers” and Evaluate whether the two rows have the same Cust_Id.Return these rows of Cust_Id. It has 3 customers and 4 sales. So, it performs this comparison a total of 3*4 or 12 times. Only 3of these comparison results in a matching.

Sort-Merge Join Algorithm [1]:-Sort merge algorithm are considered as an efficient join algorithm when compared to Nested loop join algorithm. Sort merge join algorithms have two operations and they are sorting and merging. In sorting operation, the two tables to be joined are sorted in ascending order. In merging operation, the two sorted tables are merged. Sort records of table a based on the join attribute Sort records of table B based on the join attribute.

```
Let i = 1 and j =1
Repeat
   Read record A (i)
   Read record B (j)
If join attribute A (i) < join attribute B (j) Then
     i++
Else
  If join attribute A (i) > join attribute B (j) Then
     j++
  Else
Put records A (i) and B (j) into the Query.
```

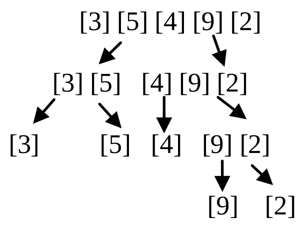


Figure 2. Example of SMJ.

Hash Based join algorithm [1]: - In hash based join algorithm, hashing and probing are the two processes. A hash table is created by hashing all records of the first table using a particular hash function. Records from the second table are also hashed with the same hash function and probed. If any match is found, the two records are concatenated and placed in the query result. A decision must be made about which table is to be hashed and which table is to be probed. Since a hash table has to be created, it would be better to choose the smaller table for hashing and the larger table for probing. The hash join algorithm is given as

```
Let H be a hash function
        For each record in table B
        Read a record from table B
        Hash the record based on join attribute value
        using hash function H into hash table
        For each record in table A
        Read a record from table A
        Hash the record based on join attribute value
        using H
        Probe into the hash table
        If an index entry is found then
        Compare each record on this index entry
        With the record of table S
        If matched then
        Put the pair into query.
```

Example:-

Consider schema of two tables Emp_Master and Emp_Info
Create Table Emp_Master (Id int, Name varchar (10), Designation varchar (10), Dept varchar (10))
Insert Emp_Master values (1,'John','Lecturer','Mech')
Insert Emp_Master values (2,'Jack','Professor','Extc')
Insert Emp_Master values (3,'Jill','AP','Comp')
Create Table Emp_Info (Id int, Dt_of_Joining Date Time)
Insert Emp_Info values (2, 13/10/04)
Insert Emp_Info values (1, 12/11/05)
Insert Emp_Info values (2, 13/10/04)
Insert Emp_Info values (3, 10/09/05)
Insert Emp_Info values (1, 12/11/05)
Insert Emp_Info values (2, 13/10/04)
Insert Emp_Info values (2, 13/10/04)

Query is written as:-Select Id int, Name varchar (10), Designation varchar (10), Dept varchar (10) From Emp_Master inner join Emp_Info on Emp_Master. Id = Emp_Info. Id Order by Emp_Info. Dt_of_Joining desc.

Existing work on adaptive join algorithms can be classified in two groups:-hash based join and sort based [3] join. Examples of hash based algorithms are XJoin, Hash Merge join, Progressive Merge join.

Double Pipelined Hash Join (DPHJ) [5]:- The double Pipelined Hash Join (DPHJ) is another extension of the symmetric hash join algorithm [7]. DPHJ has two stages. The first stage is similar to the in-memory join in the symmetric hash join and XJoin[5]. In the second stage, pairs that are not joined together in the first phase are marked and are joined in disk. DPHJ [6] is suitable for moderate size data, but does not scale well for large data sizes.

XJoin [8]:- It is a non-blocking join operator, which has a small memory footprint, allowing many such operators to be active in parallel. XJoin is optimized to produce initial results quickly and can hide intermittent delays in data arrival by reactively scheduling background processing. It show that XJoin is an effective solution for providing fast query responses to user even in the presence of slow and bursty remote sources.

MJoin [9]: - The basic idea of the MJoin algorithm is simple: generalize the symmetric binary hash join and the XJoin algorithms to work for more than two inputs. Our primary goal is to maximize the output rate during the memory-to-memory phase of the MJoin. In MJoin, the disk to-memory phase is intended to allow the system to generate outputs while its inputs are blocked, while the disk to-disk phase is intended to generate any final answers after the inputs have terminated. Interestingly, for the MJoin, how we handle memory overflow determines the output rate of the memory-to-memory phase.

Progressive Merge Join [10]: - PMJ is the adaptive non-blocking version of the sort merge join algorithm. It splits the memory into two partitions. As tuples arrive, they are inserted in their memory partition. When the memory gets full, the partitions are sorted on the join attribute and are joined using any memory join algorithm. Thus, output tuples are obtained each time the memory gets exhausted. Next, the partition pair (i.e., the bucket pairs that were simultaneously flushed each time the memory was full) is copied on disk. After the data from both sources completely arrives, the merging phase begins. The algorithm defines a parameter F, the maximal fan-in, which represents the maximum number of disk partitions that can be merged in a single "turn". F/2 groups of sorted partition pairs are merged in the same fashion as in sort merge. In order to avoid duplicates in the merging phase, a tuple joins with the matching tuples of the opposite relation only if they belong to a different partitition pair.

Hash Merge Join [11]: - HMJ is a hybrid query processing algorithm combining ideas from XJoin and Progressive Merge Join. HMJ is a new non-blocking join algorithm that deals with data items from remote sources via unpredictable, slow, or bursty network traffic. The HMJ algorithm is designed with two goals in mind: (1) Minimize the time to produce the first few results, and (2) Produce join results even if the two sources of the join operator occasionally get blocked. The HMJ algorithm has two phases: The hashing phase and the merging phase. The hashing phase employs an in-memory hash-based join algorithm that produces join results as quickly as data arrives. The merging phase is responsible for producing join results if the two sources are blocked.

***Rate based Progressive Join [12]:* -** RPJ is the most recent and advanced adaptive join algorithm. It is the first algorithm that tries to understand and exploit the connection between the memory content and the algorithm output rate. During the online phase it performs as HMJ. When memory is full, it tries to estimate which tuples have the smallest chance to participate in joins.

In this work, we used RPJ (Rate-based Progressive Join), which continuously adapts its execution according to the data properties (e.g., their distribution, arrival pattern, etc.). RPJ utilizes a novel flushing algorithm which is op-timal among all possible alternatives (based on the same statistics about data distributions, arrival patterns, etc.), and significantly enhances the efficiency of the memory-memory stage. Furthermore, RPJ maximizes the output rate by invoking the memory-disk and disk-disk in a strategic order, i.e. the next stage selected for execution is the one expected to produce the highest output rate.

New join Technique:

***DINER:* -** In this work, First algorithm is Double Index Nested-Loop Reactive join (DINER), an adaptive two-way join algorithm. DINER [4] combines two key elements: an intuitive flushing policy that aims to increase the productivity of in-memory tuples in producing results, and a novel reentrant join technique that allows the algorithm to rapidly switch between processing in-memory and disk-resident tuples, thus, better exploiting temporary delays when new data are not available. **.**

Consider two finite relations RA and RB, which may be stored at potentially different sites and are used to our local system. Incoming tuples from both relations share the available memory. A separate index on the join attribute is maintained for the memory resident part of each input relation .A separate index is maintained for the memory resident part of each input relation. When incoming tuples arrive, compute the memory occupied by the buffer and in-memory tuples being processed by the algorithm**.** Each relation is associated with a disk partition, which stores the tuples from the relation which do not fit in the memory and have been flushed to disk. Every pair of tuples between the two relations will be joined exactly once and produced result. Each tuple is residing in main memory has a join bit, which is actually part of the index. This is initially set to 0. Whenever an in-memory tuple produce a join result, its join bit set to1. [2]

***MINER:* -** Multiple Index Nested Loop Reactive join (MINER) is used for optimization of data in homogeneous environment .our experiments using two tables demonstrate that MINER [1] outperforms previous adaptive join algorithms in producing result tuples at a significantly higher rate, while making better use of the available memory. Our experiments also show that in the presence of multiple inputs, MINER manages to produce a high percentage of early results. . Consider two finite relations. It maintains for each joined relation a separate index on each join attribute. When a new tuple arrives, it is stored and indexed in the memory space of its relation, based on the relations join attribute. This new tuple that need to be joined with all the matching in-memory tuples belonging to all other relations participating in the joins. The flushing policy is applied, when memory gets full.

Difference between DINER and Existing algorithm:

a. DINER supports equi-joins and range queries.PMJ also supports range queries but it has some limitation due to its poor blocking behavior.
b. DINER will introduces flushing policy, is used to create and maintained three overlapping value regions.
c. DINER will introduces a more responsive phase that allows the algorithm to quickly move into processing tuples when both data sources block.
d. In Leaner Algorithm, DINER improves its relative performance compared to the existing algorithm.[1]

SYSTEM ANALYSIS AND DESIGN

Analysis:

Execution of Join Statements:-

To choose an execution plan for a join statement, the optimizer must make these interrelated decisions:

a. Access Paths: **-** As for simple statements, the optimizer must choose an access path to retrieve data from each table in the join statement.
b. Join Method: **-** To join each pair of row sources, any database must perform a join operation. Join methods include nested loop, sort merge and hash joins.
c. Join Order: **-** To execute a statement that joins more than two tables, SQL joins two of the tables and then joins the resulting row source to the next table. This process is continued until all tables are joined into the result.

Chooses Execution Plans for Joins:-

The query optimizer considers the following when choosing an execution plan:

a. The optimizer first determines whether joining two or more tables definitely results in a row source containing at most one row. The optimizer recognizes such situations based on Unique and Primary Key constraints on the tables. If such a situation exists, then the optimizer places these tables first in the join order. The optimizer then optimizes the join of the remaining set of tables.
b. For join statements with outer join conditions, the table with the outer join operator must come after the other table in the condition in the join order. The optimizer does not consider join orders that violate this rule. With the help of query optimizer, the optimizer generates a set of execution plans, according to possible join orders, join methods and available access paths. The optimizer then estimates the cost of each plan and chooses the one with the lowest cost.

Algorithm Overview:-

In this work, DINER algorithm is used for computing the join results of two finite relations, which is stored at different sites and are streamed to our local system and it is highly adaptive algorithm to the value distribution of the relations and potential network delays and In MINER algorithm is used for computing join results in homogeneous network environment.

Algorithm Internals:-

a. Incoming tuples from both relations share the available memory.
b. 2. A separate index on the join attribute is maintained for the memory resident part of the each input relation.
c. We used memory data structures such as, arrays, hash tables in this algorithm.
d. Compute the memory occupied by the input buffer where the incoming tuples are stored.

***Problem Definition**: -* Given two relations streamed by remote sources and under limited memory constraints, the goal is to produce the resultant join at higher rate.

Requirements Analysis:

Software Requirements:

Operating System:		Windows XP
Language	:	C#.Net
Databases	:	SQL Server 2005, MS-Access

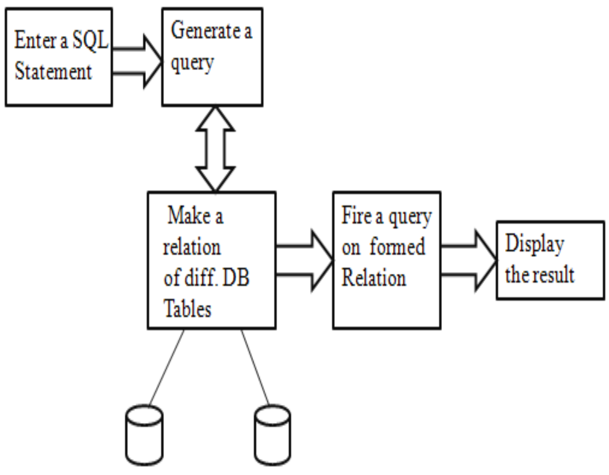
Hardware requirements:

System	:	Pentium IV 2.4 GHz
Hard Disk	:	40GB
Ram	:	1 GB
Floppy Drive	:	1.44 MB

***Design**:*

Proposed Design:-

In this work, we have considered two adaptive join algorithms, first is DINER which uses heterogeneous databases and another is the MINER which uses homogeneous databases. Important feature of the DINER and MINER is first adaptive, completely unblocking join techniques that supports range join conditions. Range join queries are a very common class of joins in a variety of applications, traditional business data processing to financial analysis applications and spatial data processing.



Enter a SQL Statement
Generate a query
Make a relation of diff. DB Tables
Fire a query on formed Relation
Display the result

Figure 3. The General Flow Diagram of Proposed System

Detailed Design:-

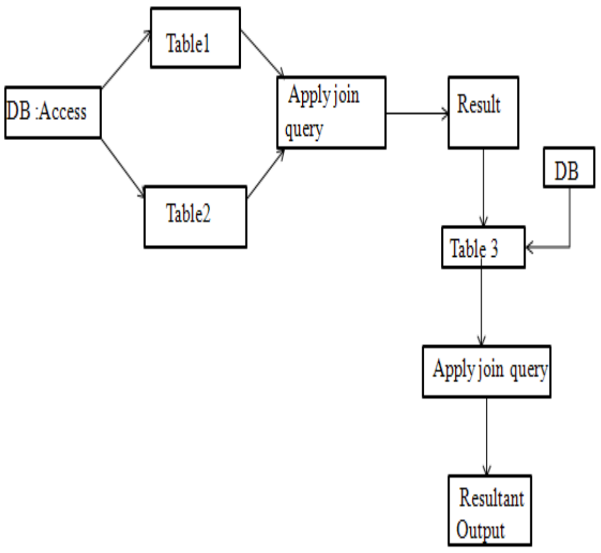


Figure 4. The General Flow Diagram of MJoin

a. Get the three tables from single database
b. Apply join query on A and B
c. If (A join with B)
d. Then (result is stored in the First_ join table)
e. This First_Join table is stored in the database
f. After that this First_Join table join with the third table
g. The result will be display

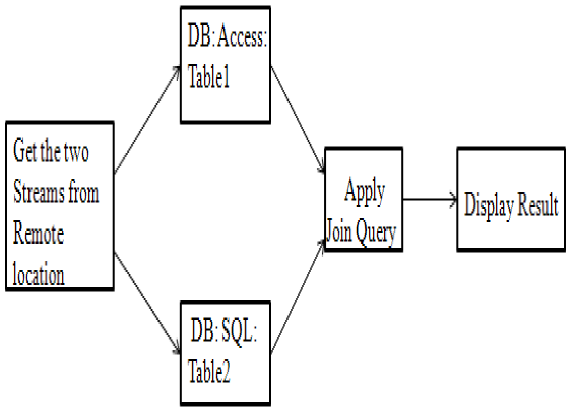


Figure 5. The General Flow Diagram of DINER

a. Get the two tables from two different databases say, A and B
b. If one relation is completely available in the buffer.
c. Then (A join with B)
d. Else
e. Wait for full relation
f. if (memory-resident tuples exist)
g. tries to join according to their matching
h. Comparing between two tables.
i. if not (flush to disk when memory becomes overflow)

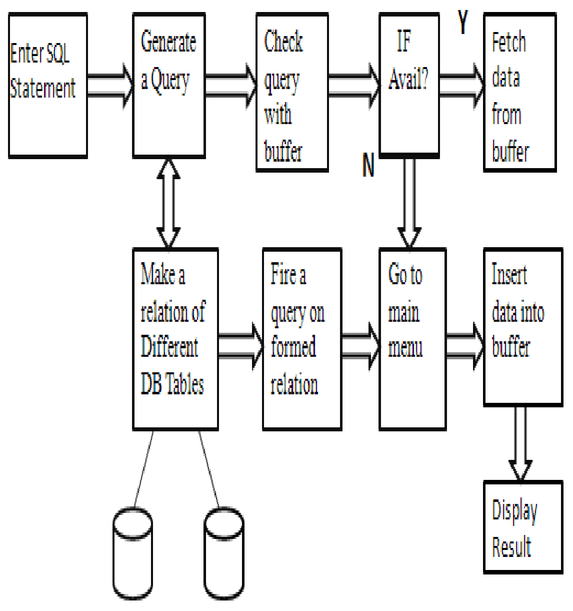


Figure 6. The General flow Diagram of MINER

a. Consider two tables A and B
b. Table A has two columns and table B has two columns
c. Table A and B contains two rows each.
d. If (query is available in buffer)
a. Then
e. Fetch index no. of that query which is already exist in the buffer
f. Display the result
a. Else
g. Go to Main menu
h. Put the (data/query) into Buffer
i. Display the Result and
j. Show the Resultant Time

SYSTEM IMPLEMENTATION

Implementation:-

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus, it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and it's constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

MJoin Description:-

a. Consider two (tables) relations, A (first table) and B (second table), C (first _join), D (third table).
b. Join A with B according to their matching in both the tables and Display the result.
c. This result is stored in C table.
d. After that, C (first_join) table is joins with D (Third table)
e. Display the result.

DINER Description:-

a. Consider two (tables) finite relations, which may be stored at potentially different sites and are used to our local system. (using two different databases)
b. It uses SQL database for one table and MS-Access database for another table.
c. Incoming tuples from both relations share the available memory.
d. Select columns from one table and it is matching with the another table. Create and select join condition according to their matchings.
e. Query is created and displayed on the screen.
f. According to column selection, Displayed one table and then another table and displayed join between two tables by using inner join.
g. Resultant time will be displayed.

MINER Description:-

a. Consider two finite relations.
b. It maintains for each joined relation a separate index on each join attribute
c. When a new tuple arrives, it is stored and indexed in the memory space of its relation, based on the relations join attribute.
d. This new tuple that need to be joined with all the matching in-memory tuples belonging to all other relations participating in the joins.
e. The flushing policy is applied, when memory gets full.
f. The optimization of MINER algorithm is achieved, when data is fetch from the buffer and shows its resultant time
g. Otherwise, go to the main menu and put the query.
h. Display the result.

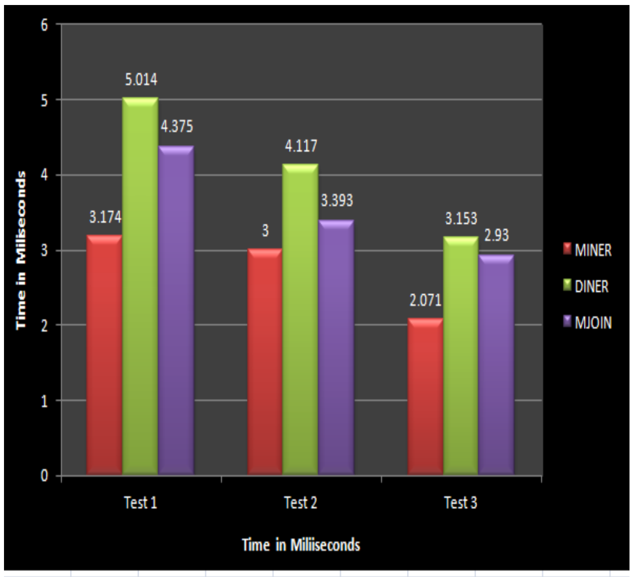
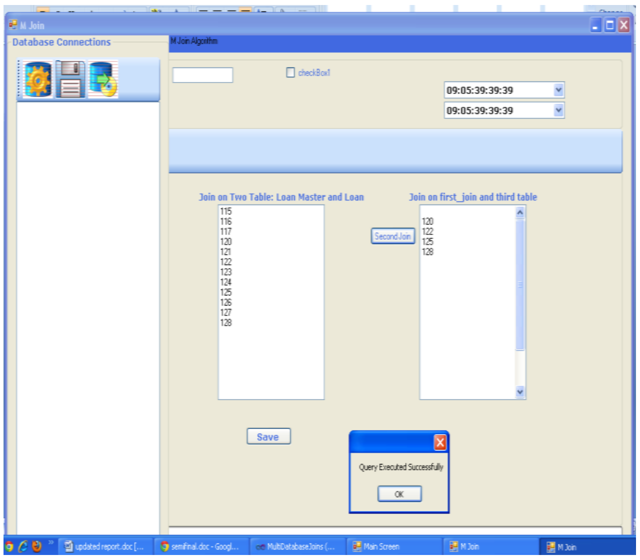


Figure. 7: Graphical Representation of MJoin, DINER and MINER

System Execution Detailed:-

a. MJoin: -

Step 1. When click on button "Open Database Connection"
Step 2. Join between two tables Loan and Loan Master which is stored in the database.
Step 3. Result of Loan and Loan Master is stored in the table "First_join".
Step 4. When click on button "second join". It shows joining between three tables.

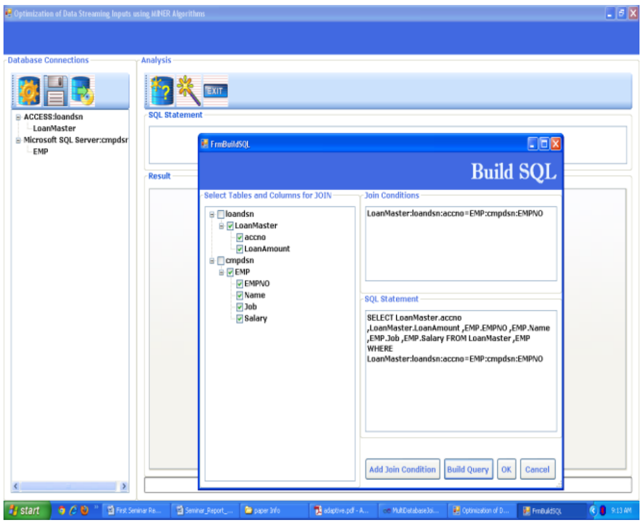


```
Join on Two Table: Loan Master and Loan     Join on first_join and third table
115                                          120
116                                          122
117                                          125
120                                          128
121
122
123
124
125
126
127
128

Query Executed Successfully
```

Screenshot No. 1: Three table join using MJoin

b. DINER:-



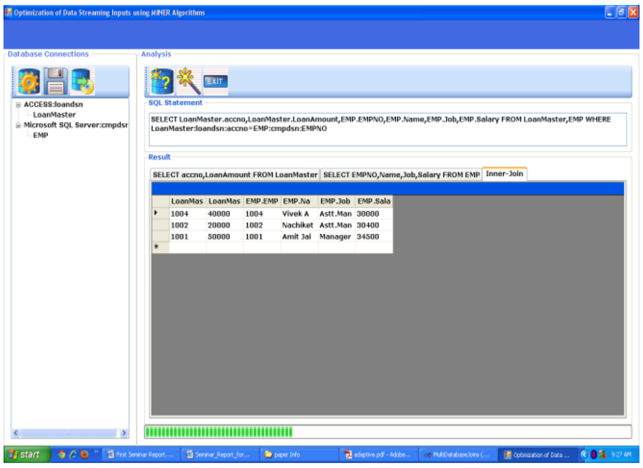
```
Join Conditions
LoanMaster:loandsn:accno=EMP:cmpdsn:EMPNO

SQL Statement
SELECT LoanMaster.accno
,LoanMaster.LoanAmount ,EMP.EMPNO ,EMP.Name
,EMP.Job ,EMP.Salary FROM LoanMaster ,EMP
WHERE
LoanMaster:loandsn:accno=EMP:cmpdsn:EMPNO
```

Screenshot No 2: Query is created

Step 1. Query is created by using selecting the columns of two tables i.e select the multiple columns from two tables where join condition.
Step 2.The query is "Select LoanMaster.accno, EMP.EMPNO FROM LoanMaster,EMP WHERE LoanMaster:loandsn:accno = EMP:cmpdsn:EMPNO".

c. DINER:-



```
SQL Statement
SELECT LoanMaster.accno,LoanMaster.LoanAmount,EMP.EMPNO,EMP.Name,EMP.Job,EMP.Salary FROM LoanMaster,EMP WHERE
LoanMaster:loandsn:accno=EMP:cmpdsn:EMPNO
```

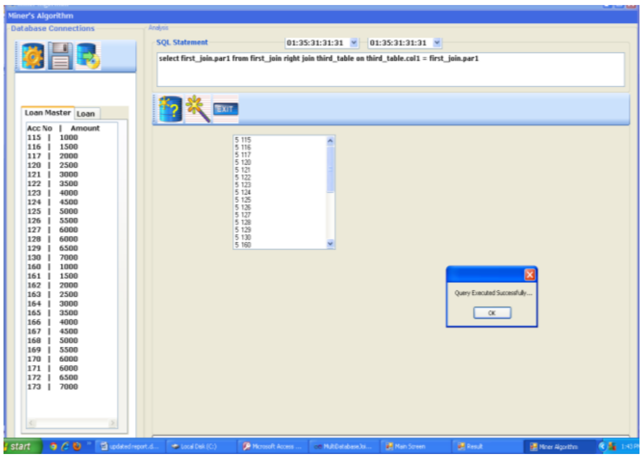
LoanMas	LoanMas	EMP.EMP	EMP.Na	EMP.Job	EMP.Sala
1004	40000	1004	Vivek A	Astt.Man	30000
1002	20000	1002	Nachiket	Astt.Man	30400
1001	50000	1001	Amit Jai	Manager	34500

Screenshot No.3: Two tables joined by using inner join

Step 1. It shows the joining between two tables' i.e LoanMaster and EMP table by using inner join

d. MINER:-

Step 1. Write the query in SQL Statement.
Step 2. Click the Button and query is executed

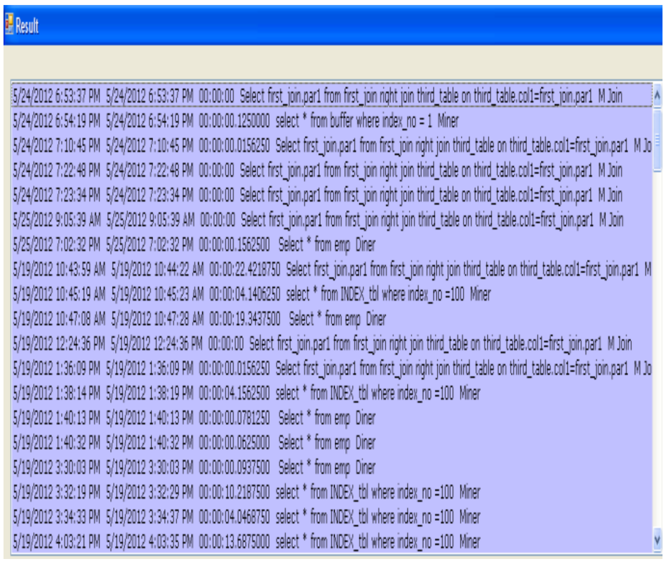


```
select first_join.par1 from first_join right join third_table on third_table.col1 = first_join.par1
```

Acc No	Amount
115	1000
116	1500
117	2000
120	2500
121	3000
122	3500
123	4000
124	4500
125	5000
126	5500
127	6000
128	6000
129	6500
130	7000
160	1000
161	1500
162	2000
163	2500
164	3000
165	3500
166	4000
167	4500
168	5000
169	5500
170	6000
171	6000
172	6500
173	7000

Screenshot No. 4: Joining according to their index

e. Result:-



```
5/24/2012 6:53:37 PM  5/24/2012 6:53:37 PM  00:00:00  Select first_join.par1 from first_join right join third_table on third_table.col1=first_join.par1  M Join
5/24/2012 6:54:19 PM  5/24/2012 6:54:19 PM  00:00:00.1250000  select * from buffer where index_no = 1  Miner
5/24/2012 7:10:45 PM  5/24/2012 7:10:45 PM  00:00:00.0156250  Select first_join.par1 from first_join right join third_table on third_table.col1=first_join.par1  M Jo
5/24/2012 7:22:48 PM  5/24/2012 7:22:48 PM  00:00:00  Select first_join.par1 from first_join right join third_table on third_table.col1=first_join.par1  M Join
5/24/2012 7:23:34 PM  5/24/2012 7:23:34 PM  00:00:00  Select first_join.par1 from first_join right join third_table on third_table.col1=first_join.par1  M Join
5/25/2012 9:05:39 AM  5/25/2012 9:05:39 AM  00:00:00  Select first_join.par1 from first_join right join third_table on third_table.col1=first_join.par1  M Join
5/25/2012 7:02:32 PM  5/25/2012 7:02:32 PM  00:00:00.1562500  Select * from emp  Diner
5/19/2012 10:43:59 AM  5/19/2012 10:44:22 AM  00:00:22.4218750  Select first_join.par1 from first_join right join third_table on third_table.col1=first_join.par1  M
5/19/2012 10:45:19 AM  5/19/2012 10:45:23 AM  00:00:04.1406250  select * from INDEX_tbl where index_no =100  Miner
5/19/2012 10:47:08 AM  5/19/2012 10:47:28 AM  00:00:19.3437500  Select * from emp  Diner
5/19/2012 12:24:36 PM  5/19/2012 12:24:36 PM  00:00:00  Select first_join.par1 from first_join right join third_table on third_table.col1=first_join.par1  M Join
5/19/2012 1:36:09 PM  5/19/2012 1:36:09 PM  00:00:00.0156250  Select first_join.par1 from first_join right join third_table on third_table.col1=first_join.par1  M Jo
5/19/2012 1:38:14 PM  5/19/2012 1:38:19 PM  00:00:04.1562500  select * from INDEX_tbl where index_no =100  Miner
5/19/2012 1:40:13 PM  5/19/2012 1:40:13 PM  00:00:00.0781250  Select * from emp  Diner
5/19/2012 1:40:32 PM  5/19/2012 1:40:32 PM  00:00:00.0625000  Select * from emp  Diner
5/19/2012 3:30:03 PM  5/19/2012 3:30:03 PM  00:00:00.0937500  Select * from emp  Diner
5/19/2012 3:32:19 PM  5/19/2012 3:32:29 PM  00:00:10.2187500  select * from INDEX_tbl where index_no =100  Miner
5/19/2012 3:34:33 PM  5/19/2012 3:34:37 PM  00:00:04.0468750  select * from INDEX_tbl where index_no =100  Miner
5/19/2012 4:03:21 PM  5/19/2012 4:03:35 PM  00:00:13.6875000  select * from INDEX_tbl where index_no =100  Miner
```

Screenshot No. 5: Output

CONCLUSION

In this work, we have successfully implemented three algorithms. The three algorithms are MJoin, DINER and MINER.

In MJoin, uses single database. During first phase of MJoin, Each new tuple is coming from one location then calculate its start time and when tuple is stored in buffer, calculate its End time and calculated difference (start time- end times) for MJoin is 3.174 ms.

In DINER, uses heterogeneous database. This algorithm can efficiently handle join predicates with range conditions, a feature unique to this technique. Each new tuple is coming from remote location. Calculate its start time, when execution started and this tuple is stored in memory then calculate its end time after that we will show the calculated (output) i.e. difference between the start time and end time for DINER is 5.014 ms.

In MINER, uses homogeneous databases. Each new tuple is coming and calculated its start time and end time. Calculates its data streaming rate is 4.375ms. But, these values for all three algorithms may be varying from time to time.

REFERENCES

[1]. J.Jayashree and C.Ranichandra "Join Algorithm for Efficient Query Processing For Large Datasets" Asian Journal of Computer Science and Information Technology 2: 3 (2012) 31 –35

[2]. Mihaela A.Bornea, Vasilis Vassalos, Yannis Kotidis, Antonios Deligiannakis: Adaptive Join Operators for Result Rate Optimization on Streaming Inputs. IEEE Trans. Knowl. Data Eng. 22(8): 1110-1125 (2010)

[3]. J. D. Ullman, H. Garcia-Molina, and J. Widom. Database Systems: The Complete Book. Prentice Hall, 2001

[4]. M. A. Bornea, V. Vassalos, Y. Kotidis, and A. Deligiannakis. DoubleIndex Nested-loop Reactive Join for Result Rate Optimization. In ICDEConf., 2009

[5]. David Taniar, Clement H.C. Leung, Wenny Rahayu, Sushant Goel. (2008) "High-Performance Parallel Database Processing and Grid Databases" A John Wiley

[6]. Z. G. Ives, D. Florescu, and et al. An Adaptive Query Execution System for Data Integration.In SIGMOD, 1999.

[7]. W. Hong and M. Stonebraker. Optimization of Parallel Query Execution Plans in XPRS. In PDIS, 1991

[8]. T.Urhan and M.J.Franklin.Xjoin: A Relatively scheduled pipilined join operator.IEEE Data Eng.Bull,23920,2000

[9]. S.D Viglas,J.F.Naughton and J.Burger.Maximizing the output rate of multiway join queries over streaming information sources.In VLDB 2003: proceeding of the 29th international

[10]. J. Dittrich, B. Seeger, and D. Taylor. Progressive merge join: A generic and non-blocking sort-based join algorithm. In Proceedings of VLDB,2002.

[11]. M. F. Mokbel, M. Lu, and W. G. Aref. Hash-Merge Join: A Non blocking Join Algorithm for Producing Fast and Early Join Results. In ICDE Conf., 2004.nal conference on very large databases 2003.

[12]. Y. Tao, M. L. Yiu, D. Papadias, M. Hadjieleftheriou, and N. Mamoulis. RPJ: Producing Fast Join Results on Streams Through Rate-based Optimization. In Proceedings of ACM SIGMOD Conference, 2005.

[13]. http://www.microsoft.com/isapi/redir.dll?prd=ie&pver=6&ar=msnhome

[14]. http://www.softwaretestinghelp.com/types-of-software-testing/