VISUAL ANALYSIS OF COST BASED DATABASE OPTIMIZERS USING PICASSO TOOL

Neeraj Sharma[1], Kavindra Raghuwanshi[2], Syed Imran Ali[3], Banshilal Patidar[4]

[1]Department of CSE, Trinity Institute of Technology and Research, Bhopal, India
neeraj.bpl83@gmail.com
[2]Department of CSE, All Saints College of Technology, Bhopal, India
mtech.kavi@gmail.com
[3]Deapartment of CSE, Trinity Institute of Technology and Research, Bhopal, India
er_imranali@yahoo.com
[4]Department of IT, Trinity Institute of Technology and Research, Bhopal, India
banshi_patidar@yahoo.com

ABSTRACT

Cost based query optimizers have seen numerous changes and up gradations since their birth in the form of System R query optimizer. Query Optimization is a process of selecting an optimal Query Execution Plan from a number of plans available for execution of query and this selection of best plan is very critical to the performance of a relational database. Picasso is a Query Optimizer analysis tool developed by the Database lab of Indian Institute of Science, Bangalore [24]. It provides graphical insights into the Query Optimization Process. With so many Database systems available in the market and each one having its own secret recipe of Query Optimization it becomes difficult to know what actually happens during query optimization. Picasso enables users to explore the world of query optimization. In this paper we briefly introduce the query optimization concept and then describe a special technique known as Plan Diagram Reduction which improves the efficiency of Query Optimization Process and makes it more Robust.

KEYWORDS

Query Optimization, Selectivity, Plan Cardinality, Plan Diagrams, Checkpoints, TPCH

1. INTRODUCTION

Structured Query Language (SQL) follows a declarative paradigm which means that the order of execution of instructions has to be determined by the SQL compiler. For this purpose the initial SQL query submitted by user is first converted into its equivalent relational algebra equation and for this expression a canonical query execution tree is generated. From this canonical tree many query execution plans or QEPs [3] can be generated using multiple techniques. Each query execution plan specifies a different order of execution of query with different set of operation. The task of Query optimizer is to search for the best possible query execution plans out of all these query execution plans. Query Optimizer plays a crucial role in determining the efficiency of Database systems. If the choice of query execution plans is not correct then the response time of the query processor will degrade. This increase in response time can be frustrating for the end user especially when the user is querying a large database in the likes of data warehouses or databases for decision support systems. Thus finding out the optimal sequence of execution becomes the overhead of database query optimizers.

DOI : 10.5121/ijcsit.2012.4120

Query processing is a multi step process. A simple query written in a declarative language such as SQL is first converted to an equivalent relational algebra expression and is then converted into a query tree which is primarily left deep or right deep trees as shown in Fig. 1[22].

Q1: SELECT Lname **FROM** EMPLOYEE, WORKS_ON, PROJECT **WHERE** Pname= 'Aquarious' AND Pnumber=Pno AND Essn=Ssn AND Bdate>'1957-12-31'

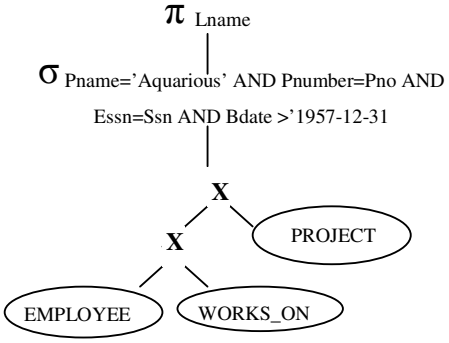

Figure 1. Query tree for Q1

Problem in query optimization is that many such Query trees can be constructed by shuffling the positions of leaf nodes and non leaf nodes or by changing the relational algebra operations (such as using a hash join instead of nested loop join). Each such query tree corresponds to a specific query execution plan.

The task of a query optimizer is to analyze all the query execution plans and select the optimal plan for executing the query. The selection for best plan is done by applying some rules (heuristics optimization) and by using some cost functions (cost based optimization). The number of query execution plans for a given plan or the plan cardinality may be huge which makes it impossible to analyze each and every plan for optimality. If much time is spent in just searching the best plan, its execution won't prove to be much beneficial. Thus a tradeoff between searching time and execution time is necessary. Because of this most query optimizers put efforts to search for the near optimal QEP instead of searching for the most optimal plan.

Another problem faced by optimizers is the selectivity of base relations. The choice for best plan is made on the basis of some complex cost functions whose major parameters are the selectivity of base relation. Since the selectivity keeps on changing frequently the cost calculation becomes wrong and a sub optimal plan may get selected. Thus a dynamic calculation of selectivity is required for getting correct value of cost functions. Static compilation of selectivity is highly prone to errors.

2. Picasso Database Query Optimizer Visualizer

Picasso [24] is a database query optimizer visualizer which generates cubical diagrams showcasing all the query execution plans that can be used for execution of a query in a specified selectivity space. Picasso tool is one of the few available Query Optimization Simulators and is written entirely in Java, is operational on database query optimizers of DB2, Oracle, SQL Server, Sybase ASE and PostgreSQL. It is already in use in academic and industrial labs used as query optimization analyzer, debugger, and for redesign aid by system developers, query optimization test-bed by database researchers, and query optimizer pedagogical support by database instructors and students.

Picasso generates a variety of diagrams that characterize the behavior of the engine's optimizer over this relational selectivity space. It generates many diagrams of which the important ones are Plan Diagram, Cost Diagram, Cardinality Diagram and Reduced Plan Diagram. However in this paper we discuss two of the most important diagrams used that are Plan Diagram and Reduced Plan Diagram.

2.1 Plan Diagrams

The *Plan Diagram* is a pictorial enumeration of the execution ***plan choices*** made by the optimizer over a relational selectivity space. The plan diagram assigns a unique color to each different optimal plan and assigns this color to all occurrences of the plan in the diagram. The diagram in Fig.2 (a) is a basic Picasso Diagram which shows 109 different plans that can be used for execution of query Q8 of TPCH Database. Each plan is represented using a different color. A plan is optimal in the area covered by its color in the plan diagram. The region of the plan diagram covered by a specific plan corresponds to the selectivities of the two base relations for which the plan will be optimal.

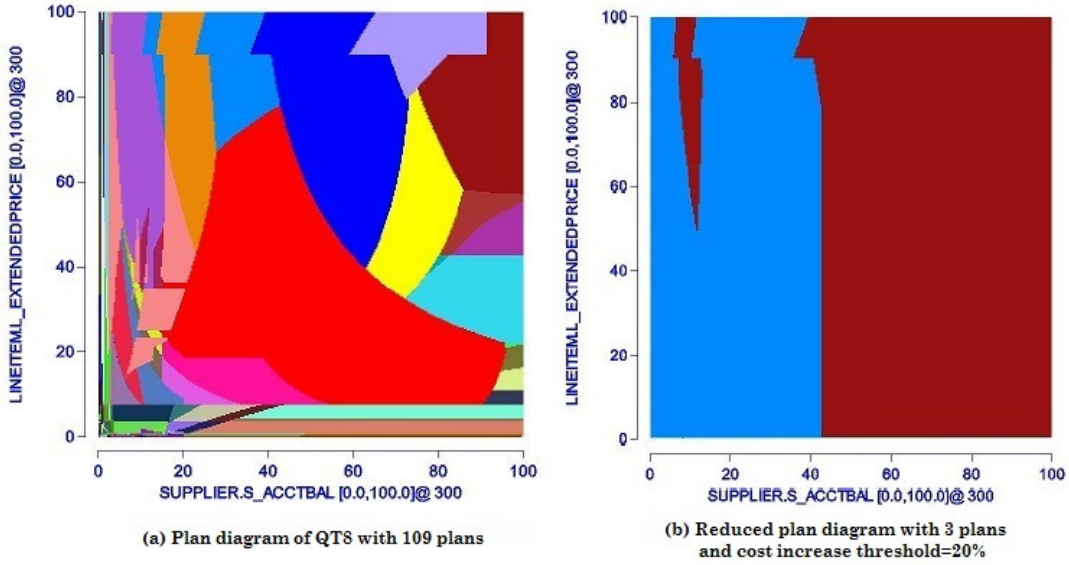

Figure 2. Plan Diagram and Reduced Plan Diagram for QT8 of TPCH

The Plan Diagram is always 2D, except when the query-template has only 1 dimension (i.e one PSP), in which case it is 1D. If the query-template has more than 2 dimensions (i.e. ≥ 2 PSPs), then the plan diagram represents 2D **slice** where the ***x*** and ***y*** axes and the constants corresponding to the rest of the dimensions.

In this paper we discuss one of the main diagrams generated by Picasso known as Reduced Plan Diagram. Reduced plan diagram shows a reduced number of query execution plans that can be used for execution of query. This reduced plan diagram can be effectively used for increasing the performance of Query Optimizer. The plan diagram in Fig. 2(a) shows 109 different plans to execute QT8 which are reduced to 3 plans in Fig. 2(b). This reduction will help a lot in increasing the performance of query optimizer by decreasing the searching time for the optimal plan and decreasing the chances of selection of wrong plan.

3. PROBLEMS IN QUERY OPTIMIZATION

Query optimization is a hard problem [4]. The selection for best execution plan is done by using many different techniques of which the prominent ones are using heuristics, some cost formulae, and randomized algorithms or genetic techniques [12]. Most Relational databases use heuristics combined with some cost formulas [22].

It is very time consuming to calculate cost of each and every execution plan to find the most optimal plan and is practically not feasible. Another problem faced by optimizers is the changing selectivity of base relations. The selection of optimal plans is based on the basis of some complex cost functions whose major parameters are the selectivities of base relations. It happens frequently that the estimated selectivities change. These wrong estimates of selectivities will lead to a poor choice of QEP.

3.1. Related Work

System R - A breakthrough work in query optimizers appeared in System R [1]. System R optimizer was the most rudimentary form of query optimizers. The cost function is based on parameters like disk access time and frequency, relation cardinality and number of tuples per page etc. The work on System R optimizer was further elaborated in [2] which discussed how the access path will be selected for execution of basic queries and for complex queries involving JOIN operations.

Eddies - The first ever technique for dynamic query optimization was discussed in [6]. The authors discuss a pure continuous adaptive query processing mechanism named Telegraph which checks for selectivities during "on the fly" time of query. Telegraph overlaps the optimization and execution phases. The initial optimal plan selected for executing a query is based on the cost functions whose parameters are determined from the system catalog. These parameters are subjected to change during run time of query. When such changes in parameters are discovered eddies try to reorder the operators used in the execution plan so as to minimize the execution cost.

Parametric Query Optimization (PQO) - System R's algorithms were modified to generate multiple optimal plans for query execution and the process was called Parametric Query Optimization or PQO [7]. In PQO multiple candidate QEPs are generated for a query, each of which is optimal for some region of the parameter space. This collection of optimal candidate plans is known as Parametrically Optimal Set of Plans or POSP. Any one plan out of these POSP is used as final QEP during run time depending on the run time values of the parameters. One significant problem with the PQO technique was that while dealing with piecewise linear functions, the solution proposed is pretty much intrusive. For this the authors of PQO suggests a modified optimization technique for Non Linear cost functions. This is known as AniPQO (Almost Non Intrusive PQO) [8].

Dynamic Optimization – Dynamic query optimization is a modern way for optimizing queries. In [4] a compile time dynamic plan optimization technique is described. Most dynamic optimization techniques use a dynamic programming model which compares cost of two plans and ignores the expensive ones. This requires a total ordering of plans. But authors of [4] suggest a "check plan" operator which maintains a partial ordering of plans.

The main problem of dynamic query optimization as pointed in [5] is the time spent in repeated collection of run time statistics like selectivities and resource availability and the searching of substitute plans. Algorithm described in [5] provides specific points in the query execution plan

and the run time statistics are collected and substitute plans are searched only at these points thus minimizing optimization time.

The most practical approach in dynamic query optimization occurs in [9] in which a generalized and practically effective technique for optimization is suggested using CHECK operators. CHECK operators were placed at specific points in a query execution plan and they check the input cardinality. If the cardinality is within the specified range the current query execution plan is retained else the plan is changed during the run time.

CHECKS increase performance but excess of checks increase the execution time. So there is a risk and opportunity trade off which makes it important to determine how many CHECKS are to be inserted and where to be inserted. Five different rules for placement of CHECK operators in the Query Execution Plan were suggested by authors.

4. PLAN DIAGRAMS REDUCTION

The biggest motivation for reducing the number of plans arises after observing the skewed distribution of plans in the plan diagrams for different queries. The amount of skewed distribution of plans can be converted into a constant value known as Gini Index [25]. Gini Index is a standard economic measure of income inequality, based on Lorenz Curve. A society that scores 0.0 on the Gini scale has perfect equality in income distribution. Higher the number over 0 higher the inequality, and the score of 1.0 (or 100) indicates total inequality where only one person corners all the income.

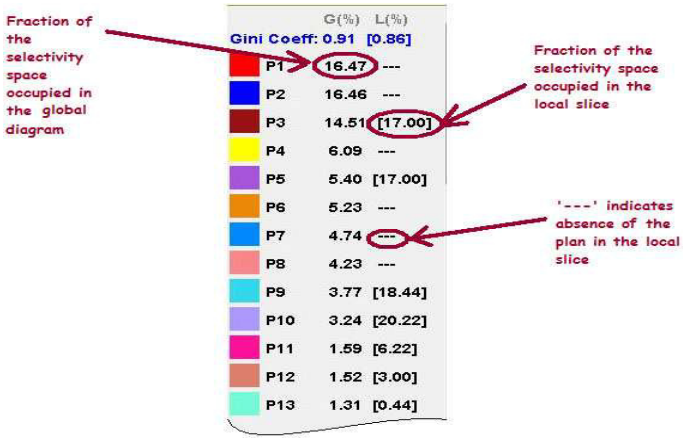

Figure 3. Gini Coefficient in the right panel of Picasso Interface

When plan diagrams are generated with Picasso tool the Gini Index values are also generated and surprisingly the Gini Index value of most of the plans diagrams was greater than 0.75 substantiating the skewed distribution of plans over the selectivity space. This is shown in table [1]. Optimizer A, B and C may refer to any of SQL Server, Oracle, DB2, Sybase or PostGre. The identities of various database systems have been kept secret so as to avoid any sort of comparison between them.

Table 1. Gini Index values for dense queries

TPC-H Query Number	OptA		Opt B		Opt C	
	Plan Card	Gini Index	Plan Card	Gini Index	Plan Card	Gini Index
2	22	0.76	14	0.72	35	0.77
5	21	0.81	14	0.74	18	0.81
7	13	0.73	6	0.46	19	0.79
8	31	0.81	25	0.72	38	0.79
9	63	0.88	44	0.70	41	0.83
10	24	0.78	9	0.69	8	0.75
18	5	0.33	13	0.57	5	0.75
21	27	0.74	6	0.80	22	0.81
Avg.	**28.7**	**0.79**	**24.5**	**0.72**	**28.8**	**0.80**

These high values of Gini Index indicate that a big percentage of plans that cover very small space in the plan diagram need not be considered for execution of query. This is because these plans are highly vulnerable to poor execution times if the selectivities change even by a small order during the execution of query. Thus we can ignore or remove these plans to be considered as a possible candidate for execution of query.

A method used to substitute these plans with other plans which have high plan area coverage is known as *Plan Swallowing* [16][17] and process wherein plan swallowing is carried on is known as plan reduction. It is defined as in [10][11]:
Given an input plan diagram P, and a cost increase threshold λ ($\lambda \geq 0$), find a reduced plan diagram R that has minimum plan cardinality, and for every plan P_i in P,

1. $P_i \in R$, or
2. $\forall$ query points $q \in Pi$, $\exists$ $Pj \in R$, such that
   $(cj(q)/ci(q)) \leq (1 + \lambda)$

That is, find the minimum-sized "cover" of plans that is sufficient to recolor P (using only the colours in LP) without increasing the cost of any re-coloured query point (i.e. whose original plan is replaced by a sibling plan) by more than cost increase threshold. Obviously, for $\lambda \rightarrow 0$, the reduced plan diagram will be almost identical to original plan diagram, whereas for $\lambda \rightarrow \infty$, the reduced plan diagram will be completely covered by a single plan.

The idea is that these smaller plans can be completely swallowed by their larger sibling plans which will effectively reduce the total number of plan cardinality in the plan diagram. There are two advantages of this approach. Firstly, it will reduce the searching time as the plan cardinality reduces and secondly, it will help us select robust plans which have higher plan space coverage and thus can tolerate higher variations in the plan selectivity. The process of plan swallowing may increase the cost of execution of a query but this increase in cost is controlled by user. This is represented as cost increase threshold (λ). Cost increase threshold (λ) of 10 is also sufficient but as proven in [14], a 20% cost increase (λ) can reduce the number of queries to an anorexic value which can significantly decrease the searching time of optimizers.

4.1. Algorithms used for Reduction

Finding an optimal solution for plan reduction is an NP Hard problem [14]. Thus the techniques used for plan reduction are predominantly heuristic in nature. Picasso uses three different algorithms for reduction. Initially COST GREEDY and AREA GREEDY were introduced in

Picasso version 1 of which COST GREEDY proved to be more efficient. In Picasso V2 a new robust plan reduction algorithm SEER (Selectivity Estimate Error Resistance) [18] was introduced which performed plan reduction and gave robust plans which can withstand run time changes in the selectivities.

Cost Greedy - This algorithm operates under the assumption that the Cost Domination Principle holds and therefore only plan swallowing possibilities in the first quadrant are considered with respect to the plan under consideration. The algorithm processes the query points starting from the top-right corner (with the Cost Domination assumption, this corner will have the highest cost of all points) and progressively makes its way to the origin, in the process using the Greedy Set Cover algorithm to recolor the points.

Cost Greedy ensure that the replacement plans are within the cost-increase-threshold at all points in the optimality regions of the replaced plans. The complexity of this algorithm is **O(mn)** where n is the number of plans in the diagram and m is the number of query points. Cost Greedy provides the best possible approximation factor with respect to the optimal reduction. Another attractive feature of Cost Greedy is that a swallowed point is re-colored *only once*, in contrast to Area Greedy where a swallowed point can be re-colored multiple times.

SEER - (Selectivity Estimate Error Reduction) – This algorithm is developed for the production of robust queries which can tolerate the changes in the selectivities of base relations during run time. These changes in selectivities can even be in orders of magnitude in real database environments [21], arising due to a variety of reasons, including outdated statistics, attribute-value independence assumptions and coarse summaries. Due to wrong selectivity estimates the performance of the replacement plan could be much worse than the replaced plan. This problem naturally leads to the concept of a **robust** replacement – that is, a replacement where the λ-threshold criterion is satisfied at all points in the selectivity space, i.e. the replacement ensures **global safety**. For this we use two implementations of SEER:

Corner Cube-SEER - CC-SEER implements a more conservative test for robust plan replacement applying Abstract-plan-costing operations at the corner hyper-cubes of the selectivity space and is therefore significantly faster than the original SEER. Moreover, its performance is resolution independent unlike SEER, and therefore the performance gap between CC-SEER and SEER increases with higher resolution diagrams. Experimental results [16] indicate that CC-SEER's reduction quality is comparable to that of SEER instead of it being more conservative.

Lite SEER - Lite Seer is a light-weight heuristic-based variant of SEER that makes its replacement decisions solely based on Abstract-plan-costing operations at the corners of the hypercube, and is therefore extremely efficient. Lite SEER is optimal in the sense that it incurs the minimum work (complexity-wise) required by any reduction algorithm. While it does not guarantee global safety, experimental results [16] indicate that in practice, its safety and reduction characteristics are quite close to that of SEER and CC-SEER.

5. EXPERIMENTAL ANALYSIS

Test bed Environment:

DATABASE AND QUERY SET

The database was created using the synthetic generator "dbgen" supplied with the TPC-H decision support benchmark [23]. It represents a commercial manufacturing environment,

featuring the following relations: REGION, NATION, SUPPLIER, CUSTOMER, PART, PARTSUPP, ORDERS and LINEITEM. A gigabyte-sized (1 GB) database was created on this schema, resulting in cardinalities of 5, 25, 10000, 150000, 200000, 800000, 1500000 and 6001215, for the respective relations.

All query templates were based on the TPC-H benchmark, which features a set of 22 queries, Q1 through Q22. Out of these 22 queries we have taken 17 queries. While plan and cost diagrams were generated for most of the queries, but to produce substantial results we focused on queries which generated *dense plan* diagrams– that is, plan diagrams whose optimal plan set cardinality is 10 or more. This is because dense plan diagrams reveal many interesting fact for analysis. For computational tractability, a query grid spacing of 30X30 and 100X100 is used.

RELATIONAL ENGINES
The relational engine used is Microsoft SQL Server 2008. Although any other relational engine could be used we prefer MS SQL Server because of its ease of availability and easier functionality.
COMPUTATIONAL PLATFORM
An Intel Core i3 CPU M370 running at 2.4 GHz with 2 GB of main memory and 240 GB of hard disk, running 64 bit version of Windows 7 Home Basic operating system, was used in our experiments. The relational engine used is Microsoft SQL Server 2008.Some changes in settings of MS SQL Server 2008 were required to allow for remote connections.

5.1 Comparison of Reduction Algorithms

We performed experimental analysis of the three algorithms and compared their performance and plan reduction efficiency. Analysis is performed in two different parts. In the first part we compare the time taken by the three algorithms for performing the reduction. Then we compare the reduction efficiency of the three algorithms.

Plan Diagram Generation Time

We first generate the plan diagrams and check the time required for generation. Table 2, 3 and 4 shows the generation time for plan diagrams of 2D queries for plot resolution of 30 and 100, and 3D queries for plot resolution of 10 and 30 respectively. These time readings were taken from the log files generated while generating the plan diagrams and are very accurate. The longest time was taken by 3D queries when executed on a plot resolution of 30. Because of limitation in the available processing power we could not execute 3D queries on a plot size of 300 as this process would take around 10-12 hours.

Table 2: Time taken for Diagram generation and No. of Plans for 2D queries, Resolution 30

Query Template	Gini Index	Resolution	No. of Plans	Time Taken in Sec.
QT2	0.73	30	23	2 min 24 sec
QT 7	0.67	30	8	2 min 30 sec
QT 8	0.79	30	24	5 min 22 sec
QT 11	0.73	30	13	46 sec
QT 16	0.94	30	27	44 sec
QT 17	0.86	30	13	36 sec

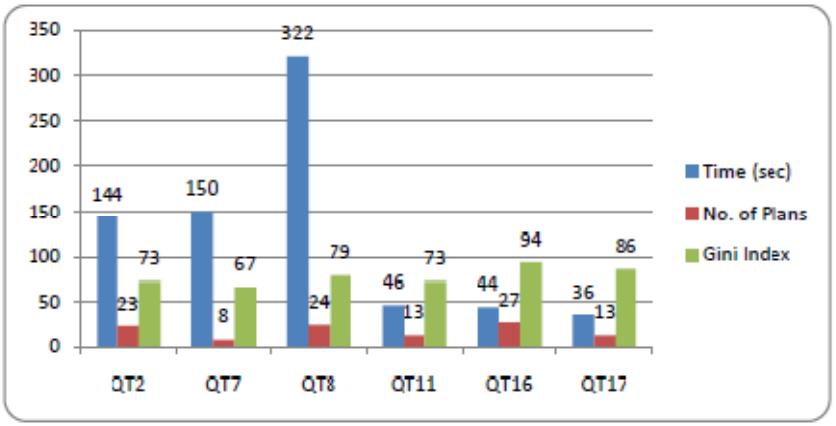

Figure 4: Comparison of Diagram Generation time for 2D queries, Resolution of 30

Table 3: Time taken for Diagram generation and No. of Plans for 2D queries, Resolution 100

Query Template	Gini Index	Resolution	No. of Plans	Time Taken in Sec.
QT2	0.72	100	36	26 min 27 sec
QT 7	0.79	100	39	1 hr 1 min
QT 8	0.73	100	16	8 min 51 sec
QT 11	0.67	100	16	27 min 47 sec
QT 16	0.94	100	31	8 min 35 sec
QT 17	0.86	100	15	6 min 56 sec

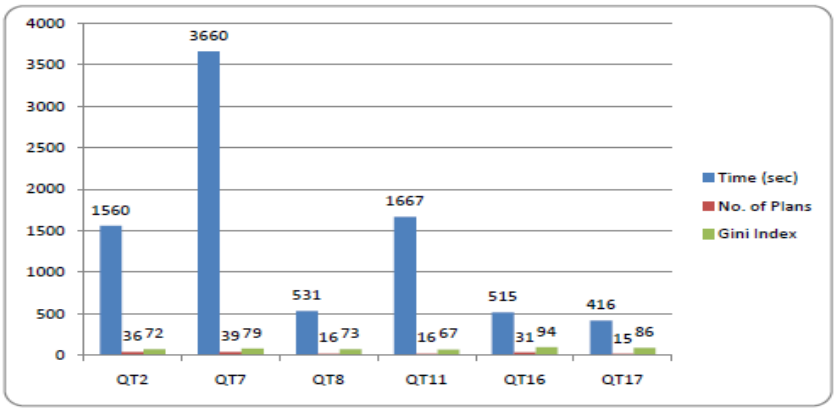

Figure 5: Comparison of Diagram Generation time for 2D queries, Resolution of 100

Table 4: Time taken for Diagram generation and No. of Plans for 3D queries, Resolution 10

Query Template	Gini Index	Resolution	No. of Plans	Time Taken in Sec.
QT 2	0.89	10	25	2 min 40 sec
QT 3	0.68	10	16	51 sec
QT 5	0.66	10	14	3 min 55 sec
QT 7	0.91	10	18	2 min 52 sec
QT 8	0.81	10	20	5 min 47 sec
QT 19	0.88	10	34	10 min 24 sec
QT 20	0.74	10	14	3 min 55sec

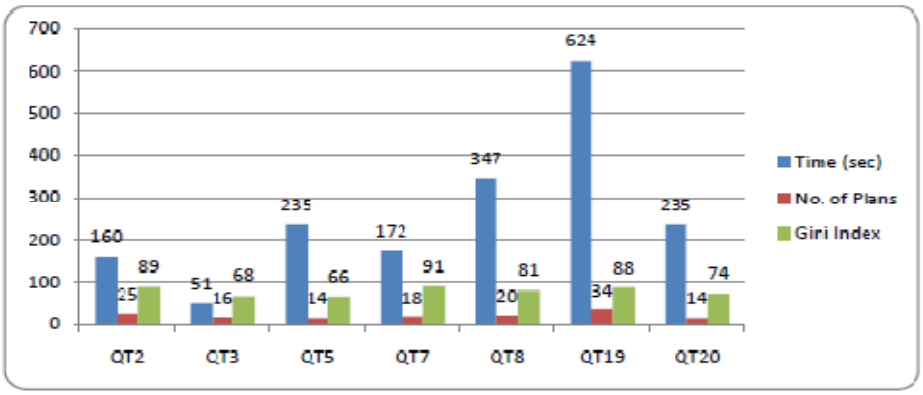

Figure 6: Comparison of Diagram Generation time for 3D queries, Resolution of 10

Table 5: Time taken for Diagram generation and No. of Plans for 3D queries, Resolution 30

Query Template	Gini Index	Resolution	No. of Plans	Time Taken in Sec.
QT 2	0.80	30	65	1 hr 14 min
QT 3	0.69	30	25	23 min
QT 7	0.91	30	25	1 hr 16 min

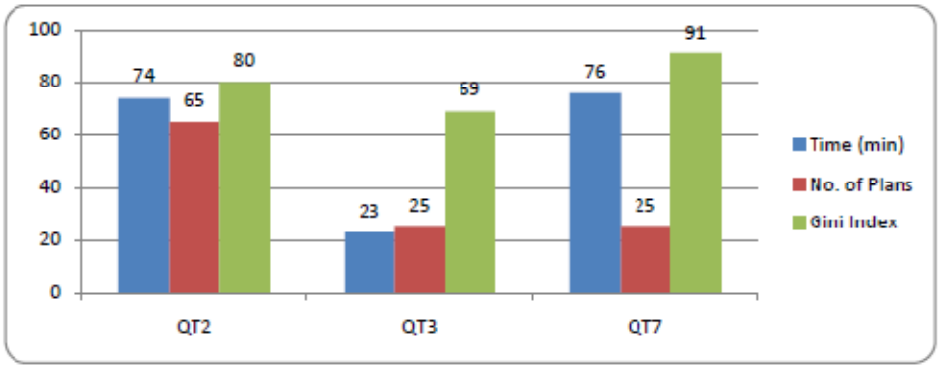

Figure 7: Comparison of Diagram Generation time for 3D queries, Resolution of 30

5.3 Comparison of Reduction Algorithms

We performed experimental analysis of the three algorithms and compared their performance and plan reduction efficiency. In the first part we compare the time taken by the three algorithms for performing the reduction. Then we compare the reduction efficiency of the three algorithms. The same Queries were used for which readings were taken in section 5.2.

5.3.1 Computation Efficiency

Table 5, 6, 7, and 8 list the time consumed for reduction of plan diagrams by the three algorithms for 2D queries with plot resolution 30 and 100 and 3D queries with plot resolution of 10 and 30.

Table 6: Plan Reduction time comparison chart for 2D queries, Resolution 30

Query Template	Plan Reduction Threshold (λ)	Plot Resolution	Time Taken by Reduction Algorithm		
			Cost Greedy	Lite SEER	CC-SEER
QT2	10	30	1 sec	28 sec	30 sec
QT 7	10	30	1 sec.	10 sec	10 sec
QT 8	10	30	1 sec	36 sec	45 sec
QT 11	10	30	1 sec.	16 sec	24 sec
QT 16	10	30	1 sec	30 sec	40 sec
QT 17	10	30	1 sec	14 sec	19 sec

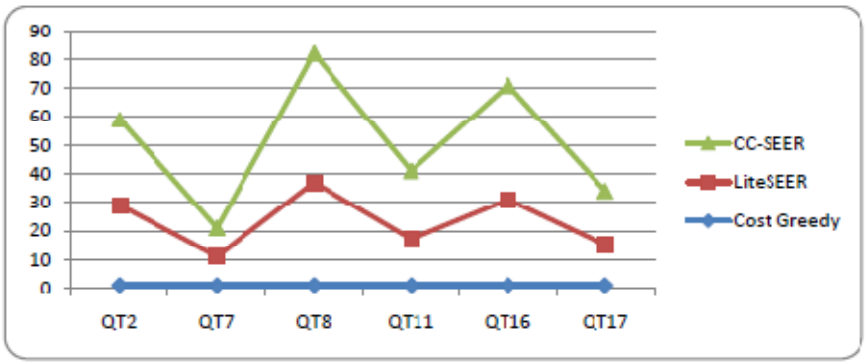

Figure 8: Comparison of Reduction Algo. for Reducing 2D Diagrams, Resolution 30

Table 7: Plan Reduction time comparison chart for 2D queries, Resolution 100

Query Template	Plan Reduction Threshold (λ)	Plot Resolution	Time Taken by Reduction Algorithm		
			Cost Greedy	Lite SEER	CC-SEER
QT2	10	100	1 sec	44 sec	1 min 20 sec
QT 7	10	100	1 sec	19 sec	22 sec
QT 11	10	100	1 sec	19 sec	19 sec
QT 16	10	100	1 sec	34 sec	47 sec
QT 17	10	100	1 sec	16 sec	23 sec

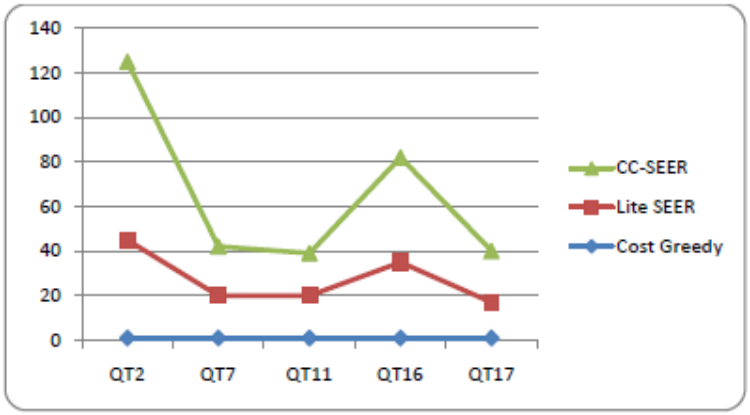

Figure 9: Comparison of Reduction Algo. for Reducing 2D Diagrams, Resolution 100

Table 8: Plan Reduction time comparison chart for 3D queries, Resolution 10

Query Template	Plan Reduction Threshold (λ)	Plot Resolution	Time Taken by Reduction Algorithm		
			Cost Greedy	Lite SEER	CC-SEER
QT 2	10	10	1 sec	35 sec	2 min 45 sec
QT 3	10	10	1 sec	17 sec	38 sec
QT 5	10	10	1 sec	18 sec	47 sec
QT 7	10	10	1 sec	21 sec	1 min 11sec
QT 8	10	10	1 sec	30 sec	1 min 35 sec
QT 20	10	10	1 sec	20 sec	45 sec

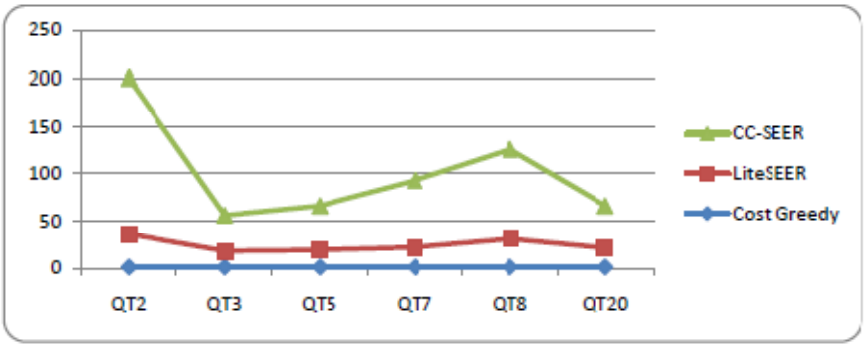

Figure 10: Comparison of Reduction Algo. for Reducing 3D Diagrams, Resolution 10

Table 9: Plan Reduction time comparison chart for 3D queries, Resolution 30

Query Template	Plan Reduction Threshold (λ)	Plot Resolution	Time Taken by Reduction Algorithm		
			Cost Greedy	Lite SEER	CC-SEER
QT 2	10	30	1 sec	1 min 31 sec	4 min 42 sec
QT 3	10	30	1 sec	28 sec	56 sec
QT 7	10	30	1 sec	30 sec	1 min 32 sec

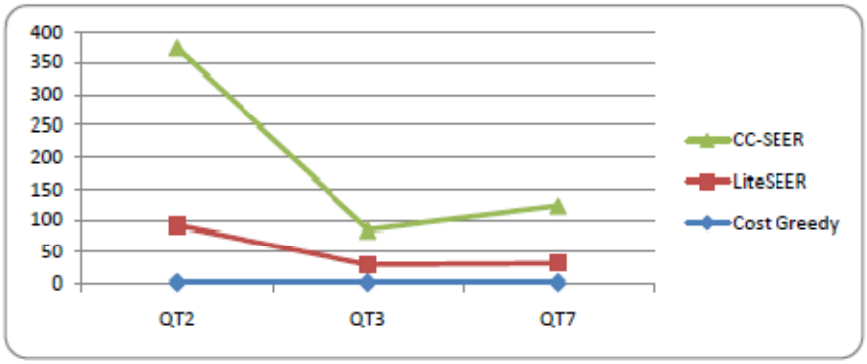

Figure 11: Comparison of Reduction Algo. for Reducing 3D Diagrams, Resolution 30

It is interesting to note that the reduction time taken by Cost Greedy algorithm was almost constant throughout the analysis for any combination of query and plot resolution. This makes Cost Greedy the fastest reduction algorithm. Next is LiteSEER which generates second best timings and then comes CC-SEER with the highest time readings. But an interesting

observation is that as the plot size increases the gap between the time taken by Lite SEER and CC SEER decreases. The large amount of time taken by LiteSEER and CC-SEER is due to the fact that these two algorithms perform extra operations so as to produce robust plans as discussed in section IV. Therefore, conceptually comparison of SEER variants with Cost Greedy is not fair. But when simple plan reductions are required, Cost Greedy scores a lot more than the SEER algorithms.

5.3.2 Reduction Efficiency

Now we check the reduction efficiency by comparing the number of plans retained after performing reduction. Again this part of analysis is carried in three parts: Table 9 and 10 lists the number of plans retained for 2D queries with plot resolution of 30 and 100 and Table 11 and 12 lists the number of plans retained for 3D queries with plot resolution of 10 and 30 respectively.

Table 10: Reduction efficiency of Reduction Algorithms for 2D queries, Resolution 30

Query Template	**Resolution**	**Original No. of Plans**	**Reduced No. of Plans**		
			Cost Greedy	Lite SEER	CC-SEER
QT 2	30	23	6 (P1-P6)	2 (P5, P8)	1 (P8)
QT 7	30	8	2 (P1, P8)	1 (P5)	1 (P8)
QT 8	30	24	1 (P11)	1 (P24)	1 (P9)
QT 11	30	13	1 (P1)	1 (P9)	1 (P4)
QT 16	30	27	22	3 (P6,P15,P24)	2 (P6,P10)
QT 17	30	13	4	1 (P4)	2 (P2, P4)

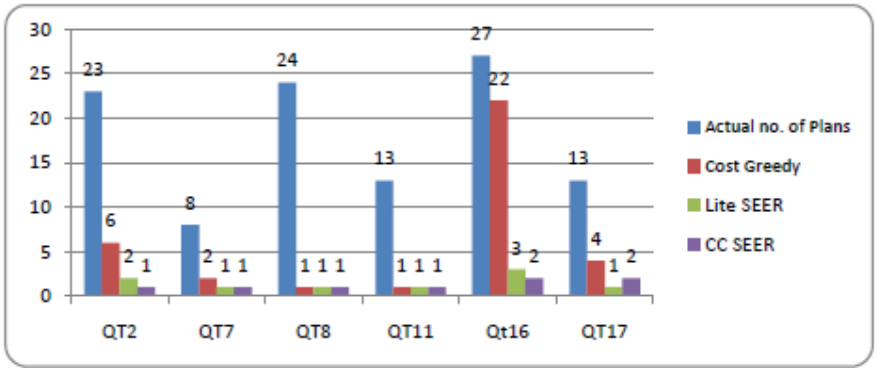

Figure 12: Reduction efficiency of Reduction Algorithms for 2D queries, Resolution 30

Table 11: Reduction efficiency of Reduction Algorithms for 2D queries, Resolution 100

Query Template	**Resolution**	**Original No. of Plans**	**Reduced No. of Plans**		
			Cost Greedy	Lite SEER	CC-SEER
QT 2	100	36	7	2 (P5, P15)	3 (P14,P20,P28)
QT 7	100	16	2 (P1, P11)	1 (P8)	1 (P11)
QT 8	100	39	1 (P13)	-	-
QT 11	100	16	1 (P1)	1 (P8)	1 (P4)
QT 16	100	31	22	5	3
QT 17	100	15	5	1 (P4)	2 (P2, P4)

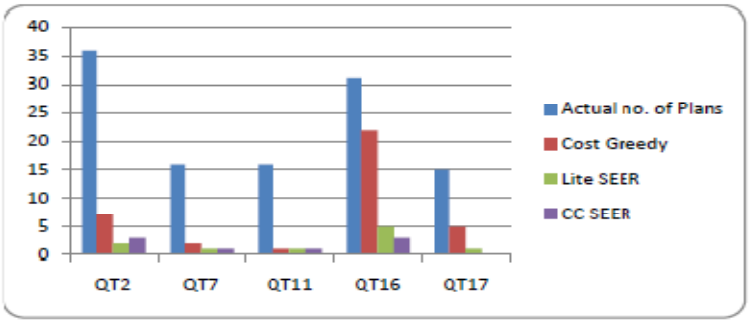

Figure 13: Reduction efficiency of Reduction Algorithms for 2D queries, Resolution 100

Table 12: Reduction efficiency of Reduction Algorithms for 3D queries, Resolution 10

Query Template	**Resolution**	**Original No. of Plans**	**Reduced No. of Plans**		
			Cost Greedy	Lite SEER	CC-SEER
QT 2	10	25	3 (P1-P3)	2 (P1, P2)	2 (P1, P2)
QT 3	10	16	8	1 (P1)	14
QT 5	10	14	2 (P1, P2)	1 (P1)	4 (P1-P4)
QT 7	10	18	1 (P1)	1 (P1)	2 (P1, P2)
QT 8	10	20	1 (P1)	1 (P1)	2 (P1, P2)
QT 20	10	14	2 (P1, P2)	1 (P1)	4 (P1-P4)

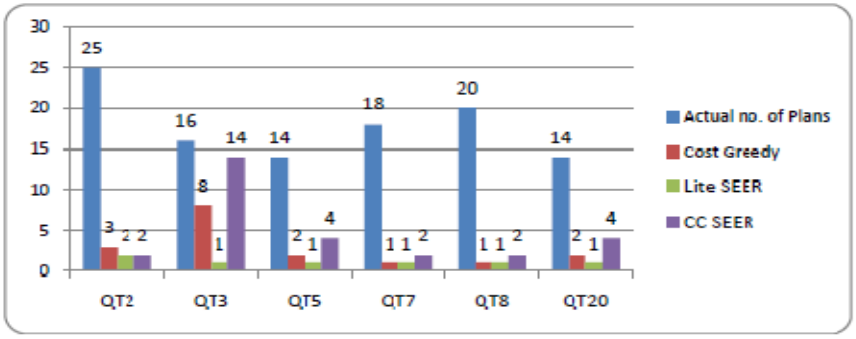

Figure 14: Reduction efficiency of Reduction Algorithms for 3D queries, Resolution 10

Table 13: Reduction efficiency of Reduction Algorithms for 3D queries, Resolution 30

Query Template	**Resolution**	**Original No. of Plans**	**Reduced No. of Plans**		
			Cost Greedy	Lite SEER	CC-SEER
QT 2	30	65	6 (P1-P6)	3 (P1 – P3)	8 (P1-P8)
QT 7	30	25	1 (P1)	1 (P1)	5 (P1-P5)
QT 3	30	25	9(P1-P9)	2 (P1, P2)	16 (P1-P16)

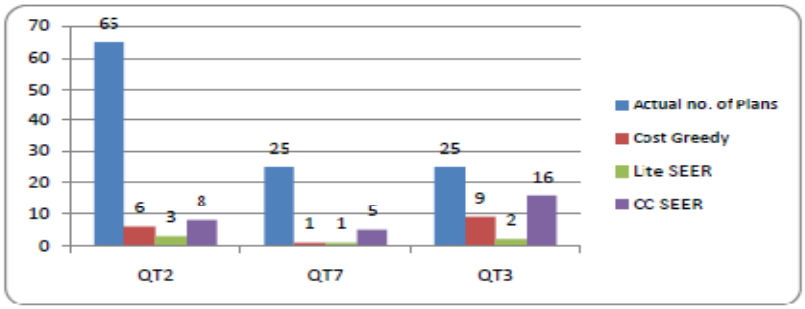

Figure 15: Reduction efficiency of Reduction Algorithms for 3D queries, Resolution 30

The observation of these three tables gives a very interesting insight into the reduction efficiency of three algorithms. The reduction patterns for 2D queries were quite different from the reduction patterns of 3D queries. The following observations were made:

- In Reduction of 2D queries, the final plans retained were different for each algorithm in most of the cases. For example, reduction of QT8, QT11. The plans retained in LiteSEER and CC-SEER were not found in Cost Greedy which proves the quality difference between the three algorithms. Thus the normal assumption about the three algorithms holds true in case of 2D queries and CC-SEER gives robust plans which mostly are missing from the output of Cost Greedy and many times from LiteSEER as well.
- In case of 3D queries the observation was quite opposite. The plans retained in CC-SEER were also present in the list of plans retained in Cost Greedy and LiteSEER. The most extreme observation is that CC-SEER retained some plans which were initially not included in Cost Greedy and LiteSEERs list. This clearly indicates that CC-SEER is not a good choice for reduction of 3D queries because Cost Greedy and LiteSEER already produced the same diagrams in very less time.

6. CONCLUSION

Query optimization is a difficult but a critical process for the database optimizers. Picasso introduces a novice way of plan diagram reduction which decreases the search complexity of query optimizers and also produces robust plans which improves the performance and reliability of query optimizers. Three such algorithms were discussed and the performances were compared. Results proved that Cost Greedy is the best algorithm in terms of execution time but if reliability was desired then Lite SEER and CC-SEER were better. For 2D queries CC-SEER must be used in spite of its long execution time but for 3D queries LiteSEER and Cost Greedy proved better than CC-SEER.

There are many interesting future works to be carried. Cost Greedy and SEER based algorithms are purely compile-time approach and it can be used in conjunction with run-time techniques such as adaptive query processing [13] for addressing selectivity errors in the higher nodes of the plan tree. Another improvisation in the design of these algorithms can be to include the technique of CHECKS suggested in [9] which can further increase the quality of plans produced after reduction. Lastly it would be interesting to use these algorithms on the upcoming TPCE dataset and some other dataset having queries with more than 3 dimensions.

REFERENCES

[1] Astrahan, M. M. et al., “System R: Relational Approach to Database Management”, ACM Transactions on Database Systems, Vol.1, No. 2, (1976)

[2] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, T. Price, “Access Path Selection in a Relational Database Management System”, SIGMOD, (1979)

[3] Johan Christoph Freytag, "Basic principles of query optimization in relational database management systems", Proceedings of IFIP Congress, (1989)

[4] L. Cole and G. Graefe, "Optimization of dynamic query evaluation plans", SIGMOD, (1994)

[5] Kabra, N; DeWitt, D. "Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans", SIGMOD, (1998)

[6] R Avnur and J. M. Hellerstein, "Eddies: Continuously Adaptive Query Optimization", SIGMOD (2000)

[7] Hulgeri, A; Sudarshan S., "Parametric Query Optimization for Linear and Piecewise Linear Cost Functions", VLDB, (2002)

[8] A. Hulgeri and S. Sudarshan, "AniPQO: Almost Non-intrusive Parametric Query Optimization, for Nonlinear Cost Functions", VLDB (2003)

[9] V. Markl, V. Raman, D. Simmen, G. Loman, H. Pirahesh, M. Cilimdzic, "Robust Query Processing through Progressive Optimization", SIGMOD (2004)

[10] Naveen Reddy and Jayant Haritsa, "Analyzing plan diagrams of Database query optimizers", VLDB (2005)

[11] Mohammad Aslam, "Picasso: Design and implementation of a Query Optimizer Analyzer", Master's Thesis, Dept. of Computer Sci. and Automation, IISc, (2006)

[12] Riham Abdel Kader, Maurice van Keulen, "Overview of query optimization in XML Database Systems", university of twente publications, (2007)

[13] Deshpande, A; Ives, Z; Raman, V., "Adaptive Query Processing", Foundations and Trends in Databases, Now Publishers, (2007)

[14] Harish D., Pooja Darera and Jayant Haritsa, "On the Production of Anorexic Plan Diagrams", VLDB (2007)

[15] Pooja Darera,"Reduction of query optimizer Plan Diagrams", Master's Thesis, Supercomputer Education & Research Centre, IISc, (2007)

[16] Harish D., Pooja Darera and Jayant Haritsa, "Robust plans through plan diagram reduction", VLDB (2007)

[17] Harish D., Pooja Darera and Jayant Haritsa, "Identifying Robust Plans through Plan Diagram Reduction", VLDB (2008)

[18] Harish D, "SIGHT and SEER: Efficient Production and Reduction of Query Optimizer Plan Diagrams", Master's Thesis, Dept. of Comp. Sci. and Automation, IISc, July (2008)

[19] Atreyee Dey, Sourjya Bhaumik, Harish D. and Jayant Haritsa, "Efficiently Approximating Query Optimizer Plan Diagrams", VLDB (2008)

[20] Jayant Haritsa, "The Picasso Database Query Optimizer Visualizer", VLDB (2010)

[21] Jayant Haritsa, "Query optimizer plan diagrams: Production, Reduction and Applications", ICDE (2011)

[22] Ramez Elmasri, Shamkant B. Navathe. "Fundamentals of Database Systems", Fifth Edition, Addison-Wesley, (2008)

[23] Transaction Processing Council, http://www.tpc.org/tpch.

[24] Project Picasso, IISC, http://dsl.serc.iisc.ernet.in/projects/PICASSO/index.html.

[25] http://en.wikipedia.org/wiki/Gini_coefficient.

Authors

NEERAJ SHARMA is Assistant Professor in Computer Science Department of Trinity Institute of Technology and Research, Bhopal with profound interest in Database Systems.

KAVINDRA RAGHUWANSHI is Assistant Professor and MTech. course coordinator in Computer Science Department of All Saints College of Technology, Bhopal.

BANSHI LAL PATIDAR is Assistant Professor and Head in Information Technology Department of Trinity Institute of Technology and Research, Bhopal.

SYED IMRAN ALI is Assistant Professor and Head in Computer Science Department of Trinity Institute of Technology and Research, Bhopal.