# Query Optimization Using a Genetic Programming Approach

**Shahid Zaman Barbhuiya** [1], **Mrs. Gypsy Nandi** [2]

[1]Don Bosco College of Engineering and Technology, Assam Don Bosco University
*Airport Road, Azara, Guwahati - 781017, Assam. INDIA.*
*shahidpt0982219@gmail.com*

[2] Don Bosco College of Engineering and Technology, Assam Don Bosco University
*Airport Road, Azara, Guwahati - 781017, Assam. INDIA.*
*gypsy.nandi@dbuniversity.ac.in*

**Abstract:*****Query optimization has been a research hot topic since the 70's. Still today new methods are introduced to optimize queries since the problem is NP-hard in nature. There exist multiple ways to execute the same query and the search space increases exponentially with increase in complexity of queries. Even the accepted methods are inadequate to optimize present day complex queries. In this paper, we propose a model based on Genetic programming to optimize such queries. We briefly explain the functioning of Genetic Algorithms and Genetic Programming and try to establish a strong base that supports application of Genetic Programming approach to query optimization.***

**Keywords:***Query Optimization, Genetic Algorithms, Genetic Programming*

## 1. INTRODUCTION

Query is an instruction to the database system issued by an end user to obtain a required output/answer. Queries act as the sole medium for the end user to interact with the database system either to store data or retrieve data and/or information. Queries can be as simple as "Find the average salary of all the employees working in an organization" or quite complex as "Find the internet usage log of all the employees working in New York branch of an organization whose annual salary is greater than $100000". The quality of performance of a DBMS is highly determined by its ability to provide the response to a particular query within the least time possible. This activity of optimizing the query to provide required output accurately and in the least possible time is termed as Query Optimization [1,6,7,8]. But the Query Optimization is in itself a NP-hard problem. Why? Let's try to understand with the help of an example.

Suppose, a user wants to find out the names of employees working in finance department of an organization. So, he issues a query to the DBMS in the form of a high level query language such as SQL. So, the query he issues is

SELECT Name from Employees Where Department = 'Finance'

The DBMS has a module called the Parser and Translator which checks the query to identify any syntax or semantic errors. The query is then translated to low level algebraic relational expression. There can usually be two or more equivalent expressions for the same query. All equivalent algebraic expressions produce the same output but might differ in the cost required in terms of disk I/O accesses, CPU time, memory buffer and transmission time (in distributed DBMS). The above query can be translated into two equivalent algebraic expressions as follows

a). $\sigma_{\text{Department='Finance'}}(\pi_{\text{Name, Salary}}(\text{Employee}))$.

b). $\Pi_{\text{Name}}(\sigma_{\text{Department = 'Finance'}}(\text{Employee}))$.

This can also be represented in terms of trees known as parse tree or query tree. It represents the order in which query is evaluated by the Query Evaluation Engine. Figure (1) shows two equivalent query trees of the above query. These equivalent query trees are also called the Query Evaluation Plans (QEP) which are then fed to the Optimizer module of the DBMS. The task of the Optimizer is to find the best query plan which satisfies the condition

*Cost (QEP$_{Best}$) = min(Costs of all QEPs )*

That is, the QEP that utilizes the least time in providing the required output to the submitting user of the query. All the QEPs together form the search space for the optimizer.

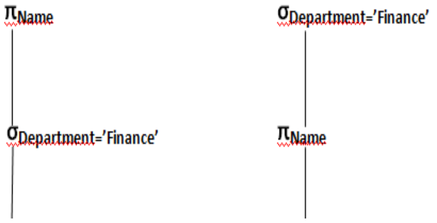

**Figure2**: Query trees corresponding to above query

We can clearly see that such a simple query as the above one has two query evaluation plans, consider the case where the required answer can only be obtained by joining records/tuples from multiple TABLEs. For N relations involved in such join queries, N! equivalent QEPs to solve the problem can be obtained by applying commutative and associative laws for joins. Figure (2) shows a graph illustrating the variation in QEPs with change in number of relations involved in join. In modern Engineering and Artificial Intelligence databases, such join queries involving large number of relations is common and hence the search space increases exponentially. Moreover, toggling between the different join algorithms and/or considering bushy trees over linear trees further increases the search space exponentially. An exhaustive or deterministic search of such a large space to find the best solution is impractical as it increases the optimization time drastically and subsequently increases the computation time of the query. A randomized search of the search space in such a scenario would be a more optimal strategy. Such a search strategy reduces the search space by random selecting QEPs and from amongst these selects a QEP which might not be the best but one with cost quite similar to the best one. Here, though the best QEP might not get selected but the search approach significantly reduces the optimization time and hence improves the computation time of the query.

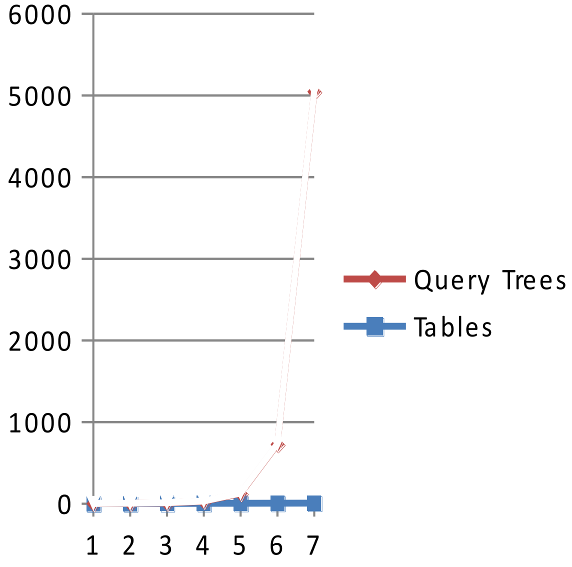

**Figure3**: Graph showing change in QEP with increase in number of relations involved in join

Evolutionary Algorithms, more specifically, Genetic Algorithms have often proved to be high performing in such NP-hard optimization problems. So, in this paper, we seek to explore the possibilities of optimizing the query by following a Genetic Programming Approach which is an extension of Genetic Algorithms.

The rest of the paper is organized such that Section 2 and Section 3 briefly describe the Genetic Algorithm and Genetic Programming paradigms respectively. In Section 4, we propose a model to apply Genetic Programming to optimize queries. Section 5 concludes the discussion with insights into future research and scopes.

## 2. Genetic Algorithm

Genetic Algorithms [2] belong to the family of Evolutionary Algorithms and are inspired by the Darwinian principle of *"Theory of Evolution"*. According to this theory, from a population of individuals, only those individuals possessing essential characteristics for survival are deemed fit to survive the competition for existence. These individuals get an opportunity to mate with other surviving individuals to produce offspring that possess the characteristics to survive and hence form the population for the next generation. Some of the individuals undergo some degree of mutation to produce possibly better (or worse) individuals to participate in the next generation. In this way, generation after generation, the individuals get better and better. Genetic Algorithms are inspired by this Evolutionary phenomenon and are believed to be very efficient in solving NP-hard optimization problems.

The set of possible solutions to the problem serve as the initial population space for the genetic algorithm. The quality of each individual is determined by a fitness function. The more fit individuals are selected to form the first generation. These first generation individuals mate with each other in the form of crossover to produce offspring that have the fit characteristics from both the parents and are expected to be more or equally fit as their parents. Crossover is generally the exchange of characteristics between two (sometimes more) individuals. Some of less fit individuals of first generation undergo mutation which signifies change in some of the characteristics of the individual. Usually, inversion of characteristics such as flipping of bits from 0 to 1 and vice-versa is what actually happens in mutation. So, by convention it is expected that weaker individuals undergo more mutation than relatively strong individuals in hope to produce an individual fitter than the one undergoing mutation. Similarly, strong individuals have higher probabilities to participate in crossover with an expectation that the offspring produced as a result of exchange of strong characteristics between parents will have fitness value greater than the parents. These new individuals generated through crossover and mutation along with some of the fittest individuals from current generation form the second generation. This cyclic process of fitness evaluation, selection, crossover and mutation continues until a stopping criterion is matched. This last generation is comprised of individuals that represent the best solutions to the given problem.

Suppose, we want to generate 8-bit long binary strings which consist of maximum number of 1's. Let's apply genetic algorithm to find some optimal solution to the problem. The steps involved in this process are as follows

a) *Encoding*: The first step is to define the structure of the individuals or possible solutions. We represent the individuals as strings of length 8. This string is also called a *chromosome* which is made up of eight bits or *genes*. Each gene either has a value of 1 or 0. Figure (3) shows two chromosomes.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

**Figure4**: Two chromosomes representing two possible solutions

b) *Fitness function*: Since our aim is to find solutions containing maximum number of 1's, so our fitness function assigns a fitness value to each of the individuals which is equal to the number of 1's in the chromosome. The chromosomes shown in Figure (3) have fitness values of 4 and 6 respectively.

c) *Selection*: There are many selection methods available which have been thoroughly discussed in [2] and [3]. For our problem, we select the chromosomes randomly from the initial population which is comprised of random 8-bit long binary strings.

d) *Crossover*: There are various crossover operators available as described in [2] and [3]. Out of these, we use the two-point crossover for illustration in our problem domain. In two-point crossover, two points (bits or genes) in the string are selected and all the bits in between these two points (inclusive) are exchanged between the two participating chromosomes to produce two offspring. Either one or both or none of the individuals are selected to participate in the next generation depending on their fitness values. Figure (4) shows the crossover between two relatively strong chromosomes of the above stated problem. The crossover produces one offspring with fitness of 8 which is greater than both its parents. Thus, we can see how crossover can result in production of fitter individuals in the process.

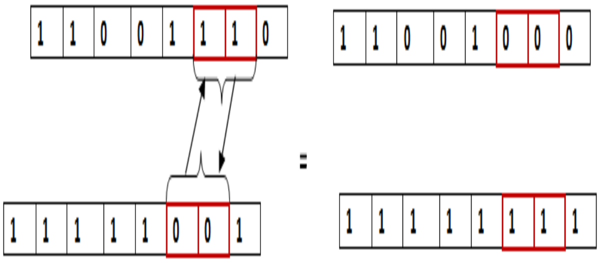

**Figure 5**: Crossover of two chromosomes to produce two offspring

e) *Mutation*: Similar to crossover, many mutation operators have also been discussed in [2]. We use the one-point mutation in our problem domain. In one-point mutation, a random bit or gene in the chromosome (string) is selected and all the bits after that bit (inclusive) are flipped or inverted e.g. 0 to 1 and vice-versa. Figure (5) shows the one-point mutation being applied to a relatively weaker chromosome to produce a stronger individual.

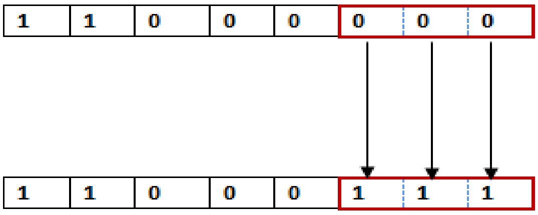

**Figure.6**: Mutation of relatively weaker individual

f) *Stopping Criterion*: The new individuals produced as a result of crossover and mutations are then selected to form the next generation population where their chances of getting selected are proportional to their fitness values. When all the individuals of the population have fitness values equal to or greater than 6, we stop the algorithm as we have reached a position where all the individuals represent the best solutions to the problem.

The above example is hypothetical and a very simple one, but it gives us an idea of how genetic algorithm functions and how it can effectively reduce the search space of a NP-hard problem. Yet, GA is still not very suited to be applied to problem domain of query optimization as the encoding of chromosomes or possible solutions in GA is in form of bit strings which may be real or binary valued whereas the most suiTABLE encoding scheme for possible solutions of query optimization problem is tree representation. In our next section, we take an overview of Genetic Programming seeking a solution to our problem.

## 3. Genetic Programming

As the name itself implies, Genetic Programming (GP) is an extension of Genetic Algorithm (GA). It follows the same cyclic process of fitness evaluation, selection, crossover and mutation of the chromosomes to optimize a NP-hard problem. The major difference between GA and GP is the encoding scheme applied to represent the chromosomes. Genetic Programming [4] was specifically designed to optimize computer programs or mathematical expressions which are difficult to represent in bit strings without distortion of the basic structure of the chromosomes (solutions). In GP, the chromosomes are represented as trees where the internal nodes of the tree represent the function(s) or operator(s) and the leaves represent the operand(s) or arguments provided to the function(s). For example, Figure (6) shows the tree representation of the mathematical expression, $x^2 + x – 5$. Here, in this expression +, -, * are the operators and hence are internal nodes of the tree. The operands x and 5 are shown as the leaves of the tree.

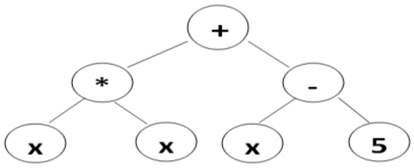

**Figure.7**: Tree Representation of $x^2 + x - 5$

After all the chromosomes constituting the initial population have been generated, the cyclic process as in GA starts over again. The fitness function assigns a fitness value to each chromosome which then participates in crossover and/or mutation to produce newer individuals for the next generation. Some of the chromosomes from the current generation may be copied to the next generation and this phenomenon is termed as reproduction.

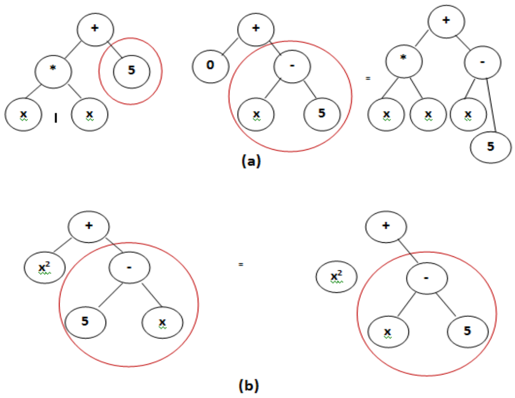

**Figure. 8:** (a) One offspring crossover between 2 chromosomes, and (b) Swap Mutation applied to a chromsome

The fitter individuals participate in crossover to produce one or two offspring. The crossover takes place by swapping of sub trees selected randomly between the parents. Some of the less fit chromosomes undergo mutation to produce new individuals which might have better fitness values greater than their respective parent. A more detailed and informative discussion on Genetic Programming can be obtained from

[4]. Figure (7) illustrates the operations of crossover and mutation in genetic programming respectively.Thus, the hindrance that was forwarded by the genetic encoding scheme used in GA to direct application of GA to query optimization problem has been efficiently eliminated by the encoding scheme applied in GP. The chromosomes in query optimization are the QEPs which can quite effectively be encoded in tree representation.

## 4. Query Optimization Using Genetic Programming (GP)

The above discussion sets a very strong inclination for genetic programming to be used for optimization of queries. GP (also GA) reduces the search space of a problem domain significantly by constructing the (sub)optimal solution(s) to the problem by combining possible good solutions and also by reconstructing (mutating) weak solutions into better solutions. As we already know that the search space of Query Execution Plans (QEPs) in the query optimization problem grows exponentially with increase in number of relations involved in a query (more specifically join queries). The best QEP can only be evaluated through an exhaustive search of this search space. But such brute force approach is impractical in situations where the search space of QEPs is quite large as this increases the optimization time drastically and hence consequently degrades performance. The solution to this is searching a portion of the search space and selecting a QEP which might not be the best one but quite close to it. But then the reduced size of the search space reduces the guarantee that the selected QEP will at least be a suboptimal one. So, we need such a method to optimize the queries which not only reduces the huge search space but also guarantees a suboptimal solution if not the optimal one which is quite close to the best one. Genetic Programming by its very nature not only reduces the search space by selecting individuals that are fit enough but also constructs an optimal solution from the suboptimal ones. So, in a sense, GP ends our quest for that one appropriate method for query optimization.

Let's dive into designing a model that uses Genetic Programming approach to optimize complex queries involving joining of multiple relations (most suiTABLE for join queries that involve more than 5-6 relations). The first requirement for the model is the population of possible solutions. Since, in our case we are trying to optimize a query, so our population intuitively consists of the different equivalent Query Execution Plans (QEPs) generated by the parser and translator. These QEPs are the query trees or parse trees and hence qualify the encoding scheme for chromosomes as defined in [3] and [4]. Figure (8) shows two equivalent QEPs for a query which involves joining of three relations A, B and C.

Such QEPs act as chromosomes for the population in Genetic Programming. We use QEP, chromosome and individual interchangeably throughout the rest of the paper. The fact that we do not have to bother about the encoding as QEPs are already in the required representation makes application of GP to the problem domain even more alluring.

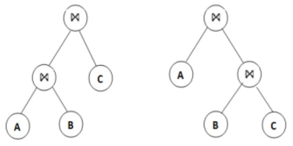

**Figure. 9:** Query trees showing join between three relations

The next step is to define the fitness function to evaluate the quality of each QEP. But before defining the fitness function, we sense a small obstacle. Since, crossover and mutation are operations that alter the structure of the tree(s) to generate new equivalent tree(s), such alteration may result in creation of a new tree that is not a valid solution to the query in observation or even might not be a syntactically correct query tree (QEP). So, to avoid such trees from getting selected into the next generation population, we introduce a condition that must be satisfied by the newly generated trees.

*Definition:* Say, if $R_1$ and $R_2$ are two relations that do not have a common attribute, then a join involving these relations results in a Cartesian Product, $R_1$ X $R_2$, creating an intermediate relation containing a copy of every record in $R_1$ against each record of $R_2$. Such operations are too costly and must be avoided as best as possible. So, if the tree generated contains a cross product, we don't allow it to participate in next generation by pruning it.

*Fitness Function:* The fitness or quality of a QEP is determined by the cost it accrues in terms of disk access time, I/O operations, CPU time, memory buffer size and network time (a special case of distributed DBMS). Out of all the factors, the disk access time is the slowest since modern day systems suffice to provide CPU time or I/O time that is quite negligible compared to the disk access time required. The disk access time is directly proportional to number of records contained in the participating relation(s). In other words, the more records a relation holds, the more time is required to retrieve the same. The memory buffer also plays a significant role in this case, as larger the buffer size; the more records can be stored in memory for subsequent processing. Moreover, the cost is further affected by the application of different selection algorithms, join algorithms and indexes as discussed in [5]. One point to notice is that the DBMS does not physically execute each and every query plan to calculate the cost; rather this cost estimation is an approximation and the DBMS maintains statistics about the relations to aid in such approximation.

Let $P_1$, $P_2$, $P_3$,…...………, $P_i$ be the generated QEPs of a query. The QEP with minimum cost is the best optimal one. Our aim is to select one such QEP which if not the best is quite similar to the best. We know, the less the cost of a QEP, the more the fitness of a QEP. If the cost of $i^{th}$ plan is given by Cost($P_i$), the fitness of this QEP or chromosome in GP is given by equation (1),

$$f(P_i) = 1 - \frac{Cost(P_i)}{\sum_{k=1}^{n} Cost(P_k)} \quad (1)$$

The value of f(Pi) is in the range of 0 to 1. Thus, equation (1) signifies that the smaller the cost of a plan, the greater it's fitness value. After defining the fitness function, we are now ready to generate our initial population for the model.

The initial population is generated by randomly selecting N QEPs from the search space. Selective pressure is a term that describes the amount of constraint we apply in selecting chromosomes to be included in our population. If selective pressure is high, weaker individuals get very little chance of selection, while low selective pressure allows weaker individuals to be selected more readily. If selective pressure is high and population size is low, then the algorithm converges too quickly and we might not get an optimal solution. On the other hand, low selective pressure and large population size though increases exploration capabilities but increases the convergence time also. We select a high selective pressure by restricting the fitness to a certain threshold. Since we select high selective pressure, we keep the population size constant throughout the entire process. We achieve such constant population by randomly selecting some QEPs from the search space in each subsequent generation along with the new individuals generated from crossover, mutation and reproduction to achieve the constant population size defined. After selecting the first generation or initial population, fitness function as given in equation (1) assigns a fitness value to each individual in the population.

*Crossover:* As discussed earlier, crossover is the process of production of offspring through exchange of parental characteristics. In GP, it is the production of one offspring or two offspring by replacement or swapping respectively of sub trees from both parent trees. Now, the question is which individuals should participate in crossover and which not. Since, it is our aim to produce stronger individuals with every new generation, we want only stronger individuals which have more fitness should participate in crossover. We define a crossover probability which is directly proportional to the fitness of the individual. The crossover probability of $i^{th}$ individual is given by,

$$Probability_{crossover}(P_i) = \gamma f(P_i) \quad (2)$$

Here, ϒ is the constant of proportionality.

Based on the crossover probability defined in equation (2), individuals are selected to participate in the crossover process. Two individuals are selected in random and subjected to two offspring crossover as discussed earlier. A point to take care of is the distortion of tree structure due to such exchange of sub trees in crossover. After crossover, the offspring produced might not even represent valid trees. So, to maintain the validity of offspring, we impose a restriction on crossover. Only similar nodes (nodes with same operator) from both trees are selected and exchanged to produce new offspring. Since, a node or sub tree is replaced by a similar one, the validity of the created offspring remain intact. The fitness values of each offspring is then evaluated using equation (1) and only if the fitness is equal to or greater than at least one parent, then the offspring is selected to possibly participate in the next generation.

*Mutation:* As discussed, mutation is the inversion of some of the characteristics. Many mutation operators have been discussed in [Koza, 1991]. We propose the swap mutation for our model. Why? Let's say there are two relations $R_1$ and $R_2$ containing 1000 and 10000 records respectively. If we join these relations on an index in the order, $R_1$ join $R_2$, the DBMS searches for a match in $R_2$ corresponding to records in $R_1$. The reverse is the case if we join both relations in the order, $R_2$ join $R_1$. Thus, it is clear the $R_1$ join R2 results in lesser searches than $R_2$ join $R_1$ and hence it can be concluded that join ordering affects the cost. So, swap mutation which swaps both the child of a node is most suiTABLE for our problem domain

After deciding on the mutation operator, we now need to decide on which individuals to apply the mutation on. By convention and nature of mutation operation, it is seen that the weaker individuals should have higher probability of undergoing mutation. So, the less the fitness of an individual, the higher is its probability to be subjected to mutation. The mutation probability of $i^{th}$ individual (QEP) is given by,

$$Probability_{Mutation}(P_i) = \frac{\delta}{f(P_i)} \quad (3)$$

Here, δ is the constant of proportionality.

Based on the mutation probability, individuals are selected to be subjected to mutation. The new individual created from such mutation is evaluated and assigned a fitness value. If its fitness is greater than the mutating individual, it is selected to possibly participate in the subsequent generation.

*Selection:* The population for the next generation is generated by combining individuals formed from crossover which accounts for 90% of new population, 5% population is composed of individuals resulting from mutation, 4% of the new population is generated by randomly selecting individuals from search space and 1% is reproduced (copied from current generation to the next). The individuals are selected based on their fitness proportionality.

*Stopping Criterion:* We define two stopping conditions for the algorithm. One condition is if we find QEP with fitness equal to or greater than 0.95. The other condition is when the best fitness value is repeated for *X* generations where *X* is user defined. This is so because we assume that since the best fitness value is repeated in subsequent generations again and again, it is most probable that this is the minimum cost to execute the given query. And hence, at the end we are left with a QEP which if not the most optimal is very close to the best one.

## 5. Conclusion

In this paper, we seek to find a solution to optimizing complex queries in the light of Evolutionary Algorithms which are quite efficient to be implemented to solve NP-hard optimization problems such as ours. We explore the query processing and optimization architectures and explore the challenges in this area. Next, we try to understand the working principle of Genetic Algorithms and its extension, Genetic Programming. We try to justify why GP is ideal for application to the problem domain. At last, we propose a model to apply genetic programming approach to optimize complex queries. The application of genetic algorithms to query optimization is still not a very explored domain and there are much possibilities in this direction. We intend to dedicate further research to improvise the proposed model as well as design a system based on the discussed model in future.

## References

[1] Chaudhuri, S. (1998, May). An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* (pp. 34-43). ACM.

[2] Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

[3] Engelbrecht, A. P. (2007). *Computational intelligence: an introduction*. John Wiley & Sons.

[4] Poli, R., & Koza, J. (2014). *Genetic Programming* (pp. 143-185). Springer US.

[5] Sethi, N., & Chauhan, M. (2013). Introduction to Query Processing and Optimization. *International Journal of Research in Computer Engineering & Electronics*, *2*(3).

[6] Wu, W., Chi, Y., Zhu, S., Tatemura, J., Hacigumus, H., & Naughton, J. F. (2013, April). Predicting query execution time: Are optimizer cost models really unusable?. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on* (pp. 1081-1092). IEEE.

[7] Park, H., Pang, R., Parameswaran, A., Garcia-Molina, H., Polyzotis, N., & Widom, J. (2013). An overview of the deco system: data model and query language; query processing and optimization. *ACM SIGMOD Record*, *41*(4), 22-27.

[8] Chaudhuri, S. (1998, May). An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* (pp. 34-43). ACM.

## Author Profile

**Shahid Zaman Barbhuiya,** is a post graduate student of Don Bosco School of Engineering and Technology, Assam Don Bosco University. He completed his Bachelors in Engineering from APIIT SD India in 2013 and currently pursuing his Masters in Technology in Computer Science and Engineering. His specialization is Data Mining.

**Mrs. Gypsy Nandi,** is currently working as Assistant Professor at Assam Don Bosco University. She completed her M.Sc and M.Phil in Computer Science. She is currently pursuing her Ph.D. in Computer Science. Her areas of interest are Data Mining and Machine Learning.