AN EFFICIENT MULTI JOIN QUERY OPTIMIZATION FOR RELATIONAL DATABASE MANAGEMENT SYSTEM USING SWARM INTELLIGENCE APPROACHES

AHMED KHALAF ZAGER ALSAEDI

A thesis submitted in
Fulfillment of the requirements for the award of the
Doctor of Philosophy in Information Technology

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

JUNE 2016
DEDICATION

To my lord Allah, my Creator teacher and master messenger, Mohamed bin Abdullah (Peace be upon him) my beloved mother, my beloved family, wife and children, all the people in my life who touch my heart, I dedicate this research.
ACKNOWLEDGEMENT

In the name of Allah, the beneficent, the merciful

I would like to express my deepest appreciation to my supervisor, Prof. Dr. Hajah Rozaida Bt. Ghazali. Without her guidance and persist help, my thesis would not have been finished. During the last few years, she has spent countless hours to patiently guide me to build interesting ideas, strengthen the algorithms and improve the writings. As a supervisor, she shows her wisdom, insights, wide knowledge and conscientious attitude. All of these sets me a good example to become a good researcher.

I would like to thank all my friends in the database group who have made my Ph.D. life more colorful. I would like to express my sincere gratitude to Prof. Dr. Mustafa Mat Deris who helps me in methodology research work and make me capable of achieving this research work.

Finally, I would like to thank my mother Rashida, without her continuous support and encouragement I never would have been able to achieve my goals and every decision I made during my Ph.D. life.
ABSTRACT

Currently, it is fairly obvious that the Multi Join Query Optimization (MJQO) is becoming the centre of attention in the context of Database Management System (DBMS). The functions consist of combination of data from multiple tables, reducing the number of needed queries, optimizing the Query Execution Plan (QEP), and moving processing abounded database servers to enhance both data integrity and performance. MJQO is an optimization task, which serves to locate the optimal QEP of a RDBMS in query processing. A major problem associated with RDBMS is the fact that they are still unable to fully meet the demands of big data. The majority of MJQO techniques encompass solution space at an extremely reduced pace. Many queries attempted to gather information from multiple sites or correlations, while every relation are compelled to answer these query via their limited resources. This lead to the access of data from many locations that are limited in their memory retention capabilities, which inevitably increase the size of the database, the number of the join, and Query Execution Time (QET). In order to eschew trapping and slow coverage difficulties in the quest to discover the optimal QEP and slow query execution time, this work proposes a total of three optimization algorithm that are based on Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Two-Phase Artificial Bee Colony (TPAPC) to solve the optimization problem in RDBMS Framework. The TPABC algorithm can be utilized to solve MJQO problems via simulation and increasing exploration and exploitation whilst balancing them for optimal results from giving queries. A directed acyclic graph, based on materialized query graph, aids in the optimization of algorithms and solving MJQO by removing non-promising QEP, which decreases the QEP combination space. Finally, experimental results demonstrate that the performance of TPABC, when compared to PSO, ACO, and native technique in the context of computational time, is very promising, which is indicative of the fact that the TPABC algorithm is capable of solving MJQO problems in shorter amounts of time and at lower costs compared to other approaches.
Sehingga kini, jelas bahawa Pengoptimuman Pertanyaan Gabungan Berganda (MJQO) telah mendapat banyak perhatian dalam bidang Sistem Pengurusan Pangkalan Data (DBMS). Fungsinya terdiri daripada gabungan data daripada jadual berganda, pengurangan bilangan pertanyaan yang diperlukan, mengoptimumkan Rancangan Pelaksanaan Pertanyaan (QEP) dan pemindahan pemprosesan pangkalan data pelayan yang banyak untuk meningkatkan integriti dan prestasi data. MJQO adalah salah satu tugas pengoptimuman, ia menggambarkan pencarian QEP yang optimum bagi DBMS dalam pemprosesan pertanyaan. Walau bagaimanapun, penyelesaian kebanyakan teknik MJQO diperoleh dalam kadar yang sangat perlahan. Oleh itu, untuk mengatasi masalah terperangkap, masalah capaian perlahan dalam pencarian QEP yang optimum dan masa pelaksanaan pertanyaan yang perlahan, kajian ini mencadagkan penambahbaikan tiga algoritma pengoptimuman. MJQO yang ditambahbaik diinspirasikan daripada Pengoptimuman Kawanan Zarah (PSO), Pengoptimuman Koloni Semut (ACO) dan dua fasa perilaku Koloni Lebah Buatan (ABC) telah digunakan untuk menyelesaikan masalah dalam Rangka Kerja RDBMS. Objektif utama kajian ini adalah untuk mengoptimumkan QEP dan mengurangkan Masa Pelaksanaan Pertanyaan (QET) dalam RDBMS dengan menggunakan pendekatan kecerdasan kawan yang diinspirasikan daripada tiga algoritma pengoptimuman, ABC, PSO dan ACO. Oleh yang demikian, Dua Fasa Algoritma Koloni Lebah Buatan yang ditambahbaik (TPABC) digunakan untuk menyelesaikan masalah MJQO dengan simulasi, peningkatan eksploitasi, mutu pencarian dan memberi keseimbangan bagi mendapatkan hasil yang optimum dengan pertanyaan yang telah ditetapkan. Struktur grafik diwakili oleh graf berkitar terarah berdasarkan kenyataan graf pertanyaan, bagi membantu algoritma pengoptimuman dalam menyelesaikan masalah MJQO, QEP yang tidak sesuai telah dipangkas, dengan itu, ia dapat mengurangkan ruang kombinasi QEP. Akhir sekali, hasil eksperimen menunjukkan bahawa prestasi TPABC berbanding PSO, ACO dan teknik naif dari segi pengiraan masa, sangat memberangsangkan dan ini menunjukkan bahawa algoritma TPABC dapat menyelesaikan masalah MJQO dalam masa yang singkat pada kos yang lebih rendah berbanding teknik lain.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter/Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE</td>
<td>i</td>
</tr>
<tr>
<td>DECLARATION</td>
<td>ii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td>ABSTRAK</td>
<td>vi</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF PUBLICATIONS</td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xiii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xvi</td>
</tr>
<tr>
<td>LIST OF SYMBOLS AND ABBREVIATIONS</td>
<td>xx</td>
</tr>
<tr>
<td>LIST OF APPENDICES</td>
<td>xxii</td>
</tr>
</tbody>
</table>

CHAPTER 1 INTRODUCTION

1.1 Research Background
1.2 Research Problems
1.3 Aims of Research
1.4 Research Objective
1.5 Significance of Research Contribution 8
1.6 Scope of Research 9
1.7 Thesis Organization 10

CHAPTER 2 LITERATURE REVIEW

11

2.1 Introduction 11
2.2 Advantages of Database Management System 11
2.3 Query Optimization 13
    2.3.1 Optimization in RDBMS 14
    2.3.2 Architecture of Query Optimizer 16
2.4 Joint Methods 19
    2.4.1 Natural Joint 20
    2.4.2 Outer Joint 21
    2.4.3 Left Joint 23
    2.4.4 Right-Outer Joint 24
2.5 MJQO in RDBMS 24
2.6 Advantage of Multi-Joint Query Optimization 25
2.7 Techniques for MJQO in RDBMS 27
2.8 Manipulation Database: SQL 29
2.9 Swarm Intelligence Overview 31
2.10 Exploration and Exploitation Properties in SI 33
2.11 Swarm Intelligent Algorithms 35
    2.11.1 Ant Colony Optimization 35
    2.11.2 Particle Swarm Optimization (PSO) 38
    2.11.3 Artificial Bee Colony Algorithm 40
    2.11.4 Behavior of Real Bees 41
    2.11.5 Modified Versions of ABC 46
2.12 Application of Swarm Intelligence in MJQO 50
2.13 Scenario leading to The Research Framework 52
2.14 Chapter Summary 56
CHAPTER 3 RESEARCH METHODOLOGY

3.1 Introduction 57
3.2 Research Methodology Framework 58
3.3 Database 60
3.4 The Design of MJQO 60
3.5 Complexity of MJQO problem 62
   3.5.1 Base line Joint Enumeration Algorithms 63
   3.5.2 Bottom-up and Enumerations 68
   3.5.3 Comparison of DPsize and DPset 69
3.6 Proposed MJQO Based on Swarm Intelligent Approaches 69
   3.6.1 Artificial Bee Colony (ABC) 70
   3.6.2 Practice Swarm Optimization (PSO) 75
   3.6.3 Ant Colony Optimization (ACO) 78
3.7 The standard Test Function Performance of SI Approaches 81
3.8 Multi-Joint Query Optimization Techniques 90
3.9 Graphical Representation 91
3.10 Multi-View Processing plan 97
3.11 Chapter Summary 102

CHAPTER 4 THE PROPOSED IMPROVE ABC ALGORITHM 103

4.1 Introduction 103
4.2 Notation 105
4.3 Artificial Bee Colony Algorithm 105
4.4 Improved ABC Algorithm for MJQO Problem 106
   4.4.1 Subset Function 107
   4.4.2 QEP Function 107
   4.4.3 Cost Estimate Function 108
   4.4.4 QET Function 109
4.5 Proposed Two Phase Artificial Bees Colony 110
   4.5.1 First phase (Employ Bee) 113
   4.5.2 Second phase (Onlooker Bee) 121
4.6 Pruning Technique 124
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7 Optimization</td>
<td>126</td>
</tr>
<tr>
<td>4.8 Chapter Summary</td>
<td>127</td>
</tr>
<tr>
<td><strong>CHAPTER 5 SIMULATION RESULT</strong></td>
<td>128</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>128</td>
</tr>
<tr>
<td>5.2 QET in RDBMS based on MySQL Server</td>
<td>129</td>
</tr>
<tr>
<td>5.2.1 Experiment One</td>
<td>130</td>
</tr>
<tr>
<td>5.2.2 Experiment Two</td>
<td>131</td>
</tr>
<tr>
<td>5.2.3 Experiment Three</td>
<td>133</td>
</tr>
<tr>
<td>5.2.4 Experiment Four</td>
<td>135</td>
</tr>
<tr>
<td>5.2.5 Experiment Five</td>
<td>137</td>
</tr>
<tr>
<td>5.3 Optimized and Unoptimized Query effect</td>
<td>139</td>
</tr>
<tr>
<td>5.4 Time Complexity</td>
<td>141</td>
</tr>
<tr>
<td>5.5 Standard Test Function Performance of SI Approaches</td>
<td>147</td>
</tr>
<tr>
<td>5.6 MJVO with proposed Optimization Algorithm Two phase</td>
<td>150</td>
</tr>
<tr>
<td>5.6.1 The effect on Number of Queries</td>
<td>151</td>
</tr>
<tr>
<td>5.6.2 The effect on Number of Data size</td>
<td>153</td>
</tr>
<tr>
<td>5.7 Effectiveness of ABC Algorithm Over Naïve Heuristic Algorithm</td>
<td>154</td>
</tr>
<tr>
<td>5.8 Efficiency of Two phase ABC Algorithm</td>
<td>156</td>
</tr>
<tr>
<td>5.9 Optimization Using TPABC Algorithm for Evaluation Time</td>
<td>158</td>
</tr>
<tr>
<td>5.10 Discussions</td>
<td>160</td>
</tr>
<tr>
<td>5.11 Chapter Summary</td>
<td>161</td>
</tr>
<tr>
<td><strong>CHAPTER 6 CONCLUSION AND RECOMMENDATIONS</strong></td>
<td>162</td>
</tr>
<tr>
<td>6.1 Introduction</td>
<td>162</td>
</tr>
<tr>
<td>6.2 Summary of Findings</td>
<td>163</td>
</tr>
<tr>
<td>6.3 Contribution of the Research</td>
<td>166</td>
</tr>
<tr>
<td>6.4 Recommendation and Future work</td>
<td>167</td>
</tr>
<tr>
<td><strong>REFERENCES</strong></td>
<td>169</td>
</tr>
<tr>
<td><strong>APPENDIX</strong></td>
<td>186</td>
</tr>
</tbody>
</table>
LIST OF PUBLICATIONS

Journals:


Proceeding:

(i) **Ahmed Khalaf Zager, Rozaida Ghazali and Mustafa Mat Deris,**” An Efficient Multi Join Query Optimization for RDBMS using Swarm Intelligent Approach proceeding in Fourth World Congress on Information and Communication Technologies , WICT 2014 (December 08-10, 2014 in Malacca, Malaysia).


(iii) **Ahmed Khalaf Zager, Rozaida Ghazali and Mustafa Mat Deris,**” Query Optimization for RDBMS using Swarm Intelligence Approaches ”, International Symposium of Information and Internet Technology”, MALTESAS conferences in 26-28 January held at Melaka, Malaysia in (2016).
LIST OF TABLES

2.1 Enroll table for natural join 21
2.2 Student table for natural join 21
2.3 Result Example Tow join 21
2.4 Student table for outer join 22
2.5 Faculty table outer join 22
2.6 Outer join for two table 22
2.7 Student table for left join 23
2.8 Faculty table for left join 23
2.9 Left join for two tables 23
2.10 Student table for Equijoin 24
2.11 Faculty table for Equijoin 24
2.12 Right outer Equijoin for two table 24
2.13 Student table for query 1 29
2.14 Result of query 1 30
2.15  Student table for query 2  
2.16  Enroll table for query 2  
2.17  Result for query 2  
2.18  Swarm intelligent techniques  
2.19  An overview of ABC based on hony bees behivoer  
3.1  Comparision of MJQO problem Based on $|P|$, $|T|$  
3.2  An example of particle Coding  
4.1  Simple notation used in this chapter  
4.2  Running example of queries and planes  
4.3  An example illustrate the QET enumerate algorithm  
5.1  Running queries for experiment one  
5.2  QET and field number for experiment one  
5.3  Running queries for experiment two  
5.4  QET and Field Number for Experiment Two  
5.5  Running queries for Experiment Three  
5.6  QET and field number for Experiment three
5.7  Running queries for experiment four  136
5.8  Query in single and multi-join to experiment four  136
5.9  Running queries for experiment five  138
5.10  QET and field number for experiment five  138
5.11  Running example of Query optimization  140
5.12  Improvement facto DPopt over DP set  141
5.13  Optimization Time in chain queris  142
5.14  Optimization Time in cycle queris  143
5.15  Time optimization for star queris  145
5.16  Time optimization for clique queris  146
5.17  The resulted obtain by PSO, ACO and ABC algorithm  47
5.18  main of best function values obtain for 50 cycle by ABC Algorithm under different colony size  148
5.19  QET based on four optimization algorithm  152
5.20  Comparison of two algorithm combined with two technique  153
5.21  QET for two phase ABC  159
## LIST OF FIGURES

1.1 Multi Join Query Optimization problem 7

2.1 Execuating SQL Queries in RDBMS 15

2.2 Sampled architcure of the query optimizatin in DBMS 16

2.3 QEP for various instancess of atemplate query 18

2.4 Swarm Intelligent Capability and Benefit 32

2.5 Ant Colony Optimization ACO 36

2.6 working Ant Colony Optimization 37

2.7 Pesudocode for ACO 37

2.8 Working of Particles Swarm Optimization 39

2.9 Pesudocode for PSO 40

2.10 Pesudocode for ABC 45

2.11 Senario leading to the reserch framework 55

3.1 Research Methodology Framework 59

3.2 Example of query graph types 61

3.3 Bottom-up order : DPSize 64

3.4 Bottom-up order : DPSet 66
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>Bottom-up enumeration : DPoptimization</td>
<td>68</td>
</tr>
<tr>
<td>3.6</td>
<td>Example of join processing tree</td>
<td>71</td>
</tr>
<tr>
<td>3.6 (a)</td>
<td>Left deep tree</td>
<td>71</td>
</tr>
<tr>
<td>3.6 (b)</td>
<td>Right deep tree</td>
<td>71</td>
</tr>
<tr>
<td>3.6 (c)</td>
<td>Bush deep tree</td>
<td>71</td>
</tr>
<tr>
<td>3.7</td>
<td>Join Operation Between two Relations</td>
<td>72</td>
</tr>
<tr>
<td>3.8</td>
<td>Simulation of the Bees Behaviour with MJQO</td>
<td>75</td>
</tr>
<tr>
<td>3.9</td>
<td>ACO for MJQO</td>
<td>80</td>
</tr>
<tr>
<td>3.10</td>
<td>Schafer Function</td>
<td>82</td>
</tr>
<tr>
<td>3.11</td>
<td>Evaluation of Mean Best Values for Schafer Function</td>
<td>83</td>
</tr>
<tr>
<td>3.12</td>
<td>Source code of schafer function</td>
<td>83</td>
</tr>
<tr>
<td>3.13</td>
<td>Sphere Function</td>
<td>84</td>
</tr>
<tr>
<td>3.14</td>
<td>Evaluation of Mean Best Values for Sphere Function</td>
<td>84</td>
</tr>
<tr>
<td>3.15</td>
<td>Source code of Sphere Function</td>
<td>85</td>
</tr>
<tr>
<td>3.16</td>
<td>Griewank Function</td>
<td>85</td>
</tr>
<tr>
<td>3.17</td>
<td>Evaluation of Mean Best Values for Griewank Function</td>
<td>86</td>
</tr>
<tr>
<td>3.18</td>
<td>Source code of Griewank</td>
<td>87</td>
</tr>
<tr>
<td>3.19</td>
<td>Rastrigin Function</td>
<td>87</td>
</tr>
</tbody>
</table>
3.20 Evaluation of Mean Best values for Rastrigin Function 88

3.21 Source code of Rosanbork Function 88

3.22 Rosanbork Function 89

3.23 Evaluation of Mean Best values for Rosanbork function 89

3.24 Source code of Rosanbork Function 90

3.25 Query Evaluation Based Swrm Intelligent Technique 91

3.26 (a) Initial Graph 92

3.26 (b) First Iteration 93

3.26 (c) Second Iteration 94

3.27 Remove invalid edges 95

3.28 Pruning Technique 96

3.29 Example Queries 98

3.30 (a) Query graph for query 1 98

3.30 (b) Query graph for query 2 99

3.30 (c) Apply MVPP merged plan for queries 99

3.31 MVPP merged plan for queries 100

4.1 Simple Examples of MJQO Based on Proposed algorithm 104

4.2 Flowchart of Proposed TPABC Algorithm 111
4.3 Phase Two with Pruning Technique

5.1 Effect Field Number for Experiment 1

5.2 Effect of Field Number for Experiment 2

5.3 Effect of Field Number for Experiment 3

5.4 Effects of Field Number on QET

5.5 Effect of Columns Number on QET

5.6 Effect of optimizing Join Queries on QET

5.7 Chain Query for Set Tables

5.8 Relative Performance of Chain Query

5.9 Cycle Queries for Set Tables

5.10 Relative Performance for Cycle Queries

5.11 Star Queries for Set Tables

5.12 Relative Performance for Star Queries

5.13 Clique Queries for Set Tables

5.14 Relative Performance for Clique Query

5.15 The Result Obtained by PSO, ACO, and ABC Algorithms

5.16 Evaluation of Mean Best Values for Schaffer Function

5.17 Evolution of mean best values for Sphere Function
5.18 Effectiveness of optimization algorithm 152
5.19 Accuracies of Different Optimization Algorithms 153
5.20 Effect of Data Size 154
5.21 Two-Combined ABC Techniques 156
5.22 Effects of Number of Queries on QET 157
5.23 Effect of Number of Relation on QET 157
5.24 Optimization Time for Two-phase ABC 159
LIST OF SYMBOLS AND ABBREVIATIONS

MJQO - Multi join query optimization
SJQO - Single Join Query Optimization
RDBMS - Relation database management system
QET - Query execution time
QEP - Query execution plane
ABC - Artificial bee’s colony
PSO - Particle swarm optimization
ACO - Ant colony optimization
NT - Native Technique
N (Ri) - Set of neighbors for a relation Ri ∈ R w.r.t. G
N(S) - Set of neighbors for a set of relations S ⊆ R w.r.t. G
Min(S) - Relation with the smallest subscript index
In a set of relations S
TPABC - Two-Phase Artificial Bees Colony Algorithm
ACOMJQO - Ant Colony Optimization for MJQO
Ci - Set of connected subsets of R with a cardinality of i
2pS^k - Set of k-way partitions of a connected subset S
pS - Set of partitions of a connected subset S
P - Set of partitions of all the connected subsets in C
T_s - Multiset of connected subsets in all partitions in PS
T - Multiset of connected subsets in all partitions in P
I_s - Set of interesting plans for a connected subset S
CSE (QEP') - Set of CSEs of a plan QET’ w.r.t. Q
Cost (QEP') - Cost of a plan QET'
JoinExp (QEP') - Join expression associated with a plan QEP
CSE (QEP) - Set of CSEs of Query Execution Plane
R = \{R_0, \cdots, R_{n-1}\} - Set of relations in Q
G = (V, E) - Query graph for Q
C = \bigcup_{i=2}^{n} C_i - Set of connected subsets of R with a cardinality of at least 2
Q = \{Q_0, \cdots, Q_n\} - Set of relations in Q
U_i = \{U_{i1}, \cdots, U_{i|u_i|}\} - Set of all the possible plans for Qi
W_i = \{W_{i1}, \cdots, W_{i|w_i|}\} - Set of all the possible plans for Qi
# LIST OF APPENDICES

<table>
<thead>
<tr>
<th>A:</th>
<th>System analysis and configuration database</th>
<th>187</th>
</tr>
</thead>
<tbody>
<tr>
<td>B:</td>
<td>Querying from Multiple Tables</td>
<td>200</td>
</tr>
<tr>
<td>C:</td>
<td>Source Code</td>
<td>207</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

3.1 Research Background

A database management system (DBMS) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and the administration of databases. Meanwhile, an RDBMS is a DBMS based on the relational aspect. As of 2015, many frequently used databases are based on the relational database model.

Multi join query optimization (MJQO) for DBMS is perhaps the most important application for searching and retrieving information in shorter amounts of time. The rapid growth in the amount of data available in the world has compelled DBMS to manage its data efficiently. This plays a big role in storage management and maintenance of the data (Wang & Strong, 1996).

Another major player in data management is information retrieval. This is the process of accessing data from relational databases, which is subsequently used to make queries into databases. On the other hand, Structured Query Language (SQL) is a programming language designed for organizing, manipulating, and retrieving data to/from RDBMS (Srivastava & Han, 2012).

A query in RDBMS can be executed via multiple approaches, where each query contains SQL clauses and filters due to a large number of alternative Query Execution Plan (QEP) being possible, making it the main difficult task when selecting optimal QEPs.
A QEP is represented as a query tree that includes information about the access method available for each relation, as all the algorithms are used in computing the relational operations in the tree. The important step is to generate codes for the selected QEP, which will then be executed in either compiled or interpreted mode to produce the query results (Singh, 2006).

In the case where the query is inserted, a query optimizer provides a large number of execution strategies that are required to analyze the data for execution by checking its validity. Hence, a large number of alternative execution plans are possible, and after a special purpose, it is not possible to analyze every possible query execution plan.

The inability to work with a large amount of data is a problem, and the major concern pertaining to this flaw is the inability to select an optimal QEP for execution. The MJQO problem appears when the number of joins in the query tree increases, which subsequently increases the number of QEP. The traditional approach is very costly and time consuming.

The problem of optimal join order in query optimization is NP-hard (Leo & Cesar, 2008). To reduce its complexity, it should be followed up with a well-accepted heuristic in RDBMS (Moerkotte & Neumann, 2006). On the other hand, (David & Frank, 2007) accounted for all bushy plans, but excluded a cross product mathematically from the enumeration space. Thus, in many case, the query optimizer ends up having to optimize for a plan that has nearly optimized.

An optimal QEP has always depended on the number of tuples used in a query. It means that the query optimizer primarily relies on statistical information to make tuple assessment, and it always depends on the accuracy of tuple assessment. Increasing the qualities of the selection process of an optimal QEP relies on additional CPU cost and increased memory consumption. Cost estimation models are mathematical algorithms or parametric equations used to estimate the costs of a QEP in terms of time or memory consumption (Dong & Shivnath, 2011).

RDBMS is the most well-known database being used nowadays, which is based on the relational database model (Leo & Cesar, 2008). Query language is an effective tool, which provides an interface to a user to store and access data. In the past few decades, SQL has emerged as a standard query language (Vidya Banu & Nagaveni, 2012); (Rashid & Ali, 2010); (Chaudhuri & Krishnamurthy, 1995).
Two components that are evident for query evaluation are the query optimizer and the query execution engine (Chaudhuri & Kr). An optimal solution should be able to evaluate the connected subset enumerate (CSE) once and reuse their results for subsequent queries to improve overall query performance. Complex multi-join queries usually take longer to evaluate due to the inherent complexity of the queries. There could be considerable performance saving by sharing the computation of CSE among the queries.

In an RDBMS context, it was shown that substantial performance saving can be obtained by using MJQO techniques. In addition to MJQO techniques in the RDBMS context, there are also some preliminary studies (Chaudhuri & Ger, 2006); (Tomasiz et al., 2010); (Lim & Herodotou, 2012) on the MJQO techniques in the DBMS context proposed by Google (Dean & Ghemawat, 2004), which have recently emerged as a new paradigm for large-scale data analysis and widely embraced by Amazon, Google, Facebook, Yahoo!, and many other companies.

There are two key reasons for this; first, the framework can be scaled to thousands of commodity machines in a fault-tolerant manner, and is thus able to use more machines to support parallel computing. Second, the framework has a simple yet expressive programming model through which users can parallelize their respective programs without being concerned about issues such as fault tolerance and execution strategy (Deng & Chain, 2014).

While all MJQO techniques (Prasad & Deshpande, 2011), (Yihong et al., 1998), (Nilesh et al., 2003) have been extensively studied in the RDBMS context, most mainly focus on optimizing a handful of SQL join queries. MJQO problem in the RDBMS context differed from these works, since the focus on optimizing a large collection (hundreds or thousands) of cross product queries produced by the applications of enumerative set-based queries.

In a traditional database, the total numbers of relations in multi-join queries are usually less than 10, which can be effectively handled by dynamic programming approaches. The complexity of this problem increases due to generation of complex multi-join queries in certain modern applications, such as knowledge-based systems, decision support systems, expert systems, Online Analytical Processing (OLAP), and data mining.

An increase in the number of tables in the join query also increases the number of alternative QEP, which complicates the optimizer’s task. Traditional methods are
not able to solve this optimization problem effectively due to the increased size of the data and larger number of tables (Dong et al., 2011). Deterministic algorithms, greedy algorithms, and heuristic algorithm-based approaches have tried to approximate the optimal solution, but their performance remains weak (Steinbrunn & Kemper, 1997).

This problem is then tied with genetic approaches and randomized approaches, such as tabu search, ant colony, bee colony, etc., all of which performs better (Kadkhodaei & Mahmoudi, 2011), but better quality performing solution is still vital. Another work has proposed a new algorithm that utilizes a cuckoo search algorithm (Yang & Deb, 2009) combined with the tabu search algorithm (Glover & Ullman, 1989) to seek better solutions and determine the optimal join order. It is an integrated part of the query optimizer. The optimizer generates a QEP, which takes some time to execute. All authors are unable to find an optimal solution to this problem due to the usage of only one database, and the results obtained were based on the only number of tables in the database, which is insufficient.

3.2 Problem Statements

In this study, there are two new problems, namely MJQO and Single Join Query Optimization (SJQO) in RDBMS. They are a crucial factor that affects the capability of the database. The MJQO technique used in RDBMS should aim to obtain results of each query efficiently, and the process of query should be optimized for time efficiency as well.

However, MJQO used in an RDBMS are inefficient in terms of Query Execution Time (QET) and cost on average. The traditional query optimization technology wasted a long time per query and for the staff when trying to request information on the work(s). This increases the daily and annual costs in institutes or company. The traditional applications of RDBMS are inefficient in terms of QET and cost. The number of joins N involved in a single query is relatively small, usually N < 10.

With the expansion of the database application, the traditional query optimization technique are unable to support some of the latest database applications, such as applications of Decision Support System (DSS), OLAP, and Data Mining (DM), which may demand a query of more than 100 genes.
When multiple users and variety queries access distributed federated database multiple tables with data variety, the tables must be joined. This can result in many database operations, leading to increased database sizes to huge tables, and join and slow processing or a deadlock situation on the other hand, queries need to return answer quickly to clients. To solve this problem, minimizing the number of joins, query plans, queries, and increased sharing are all needed in order to decrease administration time (less cost).

Hence, such shortfall in the traditional query optimization is gradually exposed. It is therefore necessary to explore new techniques to solve the MJQO problem. Since MJQO is an NP hard problem (Li Liu & Dong, 2008) with increased join, the number of QEP corresponding to a query grows exponentially, which leads to computational complexity of MJQO problem.

Hence, the need to acquire an improved quality and performance. The implications of these criteria are important to increase speed of query and reduce cost in RDBMS. Therefore, a new intelligent approach, such as the swarm intelligent approach that performs well, shorter QET, and low cost are all required.

Solving problems with a heuristic algorithm becomes a hotspot as it appears on many location or site of RDBMS, therefore needing multi-optimization or decentralized optimization, as proven in certain studies, such as ACO (Li Liu & Dong, 2008), Greedy Algorithm (GA) (Prasan & Bhobe, 2000), Genetic Algorithm (GA), ABC, (Abber & Mourad, 2013) etc. Several approaches have been proposed to model the specific intelligent behavior of meta-heuristic being applied for solving combinatorial problems.

The state-of-the-art work in this direction (Tomasz & Potamias, 2010) proposed two sharing techniques for a batch of jobs. Recent researchers have used different models to solve the MJQO problem. However, they have been unable to provide a better solution in reducing the corresponding time and cost. Traditional methods are not able to solve this optimization problem effectively due to the increased data size and large number of tables (Dong, 2008).
The optimal join order in RDBMS framework has been widely adopted by modern enterprises, such as Facebook (Thusoo & Borthakur, 2010), to process complex analytical queries on large data warehouse systems due to its high scalability, fine-grained fault tolerance, and easy programming model for large-scale data analysis. Given the long execution times for such complex queries, it makes sense to spend more time optimizing such queries to RDBMS for all processing time.

While the optimal join order problem has recently attracted much attention in a conventional RDBMS context (Kiyoshi & Guy, 1990); (Guido Moerkotte, 2006); (Guido & Thomas, 2012); (Isard & Prabhakaran, 2009); (Pit Fender & Guido, 2013); (Pit Fender & Thomas Neumann, 2012); (Fender & Moerkotte, 2012); (Roy & Siddhesh, 2000); (Nilesh & Sudarshan, 2003); (Zhou & Lehner, 2007), the developed solutions are not applicable to RDBMS due to the differences in query evaluation framework and algorithms.

The optimal join order problem in RDBMS has a larger join enumeration space compared to that in RDBMS due to the presence of multi-way joins. There has been good work in RDBMS context for complexity study (Kiyoshi & Guy, 1990); (Moerkotte, 2006); (Fender & Guido, 2012); (Fender & Neumann, 2012).

To the best of our knowledge, there has not been any prior work on the study of these problems in the presence of multi-way joins in DBMS context. First, the intermediate results in RDBMS are always materialized instead of being pipelined as in RDBMS, which simplifies the MJQO problem in two ways.

Second, the MJQO problem in RDBMS may incur deadlocks due to the pipelining framework (Nilesh & Sudarshan, 2003), while RDBMS does not have deadlock problem due to the materialization framework. Materializing and reusing results of Connected Subset Enumerate (CSE) in RDBMS may incur additional materialization and reading costs due to the pipelining framework. However, since the intermediate results always materialized in the DBMS framework, and there is no additional overhead incurred by the technique.

Although the MJQO problem in RDBMS has been shown to be a very difficult problem with a search space that is doubly exponential in the size of the queries (Prasan, & Siddhesh, 2000); (Nilesh & Sudarshan, 2003); (Jingree & Lehner, 2007), the simplification in RDBMS enables them to propose join order algorithms for the MJQO problem in RDBMS, however, they are unable to reduce the cost associated with QET, cost, and search spaces.
The large search space, number of possible plans, and many semantically equivalent logical plans, logical plans with N operators have \(2^n\) possible placement decision. In a simple example, the following figure shows different possible plans for only 3 joins on 4 tables in Figure 1.1.

![Figure 1.1: Multi Join Query Optimization Problem](image)

They share the same (A JOIN B) subtree. The existing techniques calculate the cost for all possible plans, which means it takes a long time when using swarm intelligent approaches instead of computing the cost of this subtree in every plan, compute it once, save the computed cost, and reusing it when seeing this subtree again. Using this swarm technique results in us having a \((2^N)! / (N+1)!\) time complexity, “just” 3N. In our previous example with 4 joins, it means passing from 336 ordering to 81.

### 1.3 Aim of Research

This study aims to provide a comprehensive and in-depth research for a systematic study of MJQO problem in the RDBMS paradigm and proposed swarm intelligence approaches, namely standard ACO, PSO, and improve the Two-Phase Artificial Bees Colony Algorithm (TPABC).
The proposed algorithm is used to search for and insert the query execution plan and optimal global query execution plan to solve the MJQO problem in order to RDBMS to reduce time, cost, and increase the performance of RDBMS.

1.4 Research Objectives

To achieve the research aims, the objectives are as follows:

(i) To design an MJQO for a RDBMS using a query graph based on Pruning and Materialize Techniques.

(ii) To propose a new Two-Phase Artificial Bee Colony (TPABC) by removing the scout-bee agent in order to improve the exploration factor.

(iii) To optimize Query Execution Plan (QEP) and Query Execution Time (QET) for (i) using the proposed (ii).

(iv) To compare the performance of the proposed method in (ii) with other QEP-swarm-based, such as PSO and ACO for processing time and accuracy.

1.5 Significance of Research

An important component in RDBMS is the query optimization. A user request is usually expressed in high-level, non-procedural language describing the condition produced by RDBMS’ need to satisfy.

The main problem in the RDBMS is the volume, which grows from 10 GB to 100 TB, or Exabyte in recent years. Query processing needs to be combined with non-related sources over distributed database to obtain data with huge spaces.

Each query in the optometry phase produces more than one query plan, and the optimizer tries to select the best plan at lower costs. All clients see similar views, and are able to find similar replicas of unstructured data, which leads to very expensive throughput and takes a long time for a user or client in a company, resulting in loss of income.
The multiplicity of human needs is increasing alongside limited resources, such as the MJQO problem. Economic resources are limited and insufficient to satisfy all human needs characterized by parochialism and the lack the human needs of multiple repeated renewal, such as the need to constantly include food, housing, treatment, and jobs. Multi-join query optimization problem has been widely addressed in RDBMS.

Therefore, it is necessary to design an efficient MJQO to determine the best QEP and minimizing the number of queries or objectives and joins based on a swarm intelligence approach that can be adapted to solve the MJQO problem. The proposed TPABC optimization algorithm is used to select an evaluation plan for a batch of queries and best plans in RDBMS. This is done by expanding exploration to find the optimal QEP for MJQO in order to improve the performance of RDBMS. The exploitation process is increased using TPABC to find the global optimal plane from command sub-expression queries sharing.

1.6 Scope of Research

This research aims to enhance the overall statues on MJQO in RDBMS to solve MJQO problem, which is the NB-hard problem in RDBMS. The study proposed swarm intelligence approaches, such as (ABC, PSO, ACO), as new methods to reduce the complexity and cost in order to solve this problem. All these algorithms are used to optimize QEP, QET and cost. The research work proposed TPABC to improve exploration and exploration factors to increase the performance of the database. The study attempt to solve optimal join order problem in RDBMS based on four types of query graph in RDBMS framework.

1.7 Thesis Organization

This thesis is organized and divided into six chapters. The first chapter introduces the research background, problem statements, and objectives and contributions. Chapter two presents a comprehensive literature review of the problems in RDBMS and provide an overview of the swarm intelligence-based algorithm, such as ABC, ACO, and PSO and joint techniques in RDBMS.
Chapter three encompass the methodology used to carry out the study systemically. It consists of optimization algorithm (i.e. ABC, PSO, and ACO) and two new techniques to solve MJQO problems in DBMS. Chapter four explains the proposed improve TPABC swarm-based MJQO in DBMS to solve MJQO problem, and compares TPABC with (naive heuristic algorithm) to improve factors of exploration and exploitation. Chapter five simulate the result and analysis data of both MJQO and QET. Finally, Chapter six conclude the work and provide suggestions and contribution of the research, and points out some directions for future work.
CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The second chapter of the thesis is the heart of an investigation, in which it provides an overview of contemporary literature in a broad academic and historical context (Boote & Beile, 2005). The chapter sets to describe the focus or content of the study and provide definitions of the scope of the study. This literature review explores there domain themes of the research work: Relational Database Management System (RDBMS) performance, Multi Join Query Optimization (MJQO) as good issues to improve RDBMS performance and setups swarm intelligent approaches as a technique to solve MJQO problems. The scope of this literature review is expanded to include the researches that examine the domain themes of the research work, the MJQO problem has been widely addressed in Relational Database Management Systems (RDBMS).

2.2 Advantages of Database Management System

Because data are the crucial raw material from which information is derived must have a good method to manage such data. DBMS helps make data management more efficient and effective, in particular, a DBMS provides advantages such as improved data sharing. The DBMS helps create an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment to improve data security.

In cases where more users access the data, the greater the risks of data security breaches. As such, it is noted that corporations, ensuring the corporate data are used properly by investing considerable amounts of time, effort, and money.
Therefore the use of DBMS provides a framework for better enforcement of data privacy and security policies. Better data integration with wider access that allows well-managed data are able to promote an integrated view of the organization’s operations and a clearer view of the big picture.

It becomes much easier to see how actions in one segment of the company affect other segments. Data inconsistency exists when different versions of the same data appear in different places. The RDBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a query is a specific request issued to the DBMS for data manipulation, for example; to read or update the data simply put, a query is at work, and an ad hoc query is a spur-of-the-moment work.

RDBMS sends back an answer (called the query result set) to the application. Technological advancements around transmission of data through the network, have largely influenced the cost of transmitting the data per terabyte over long distances (Gelogo & Lee, 2012). Furthermore, the RDBMS has achieved progress in two company’s dimensions: data management and data transfer.

Based on the relate research, data management happens to be more costly than data transfer (Gelogo & Lee, 2012; Garefe, 1996). In addition, there is a rapidly growing interest in outsourcing DBMS tasks to third parties that can provide these tasks for much lower cost due to economy of scale.

Designation of a new outsourcing model has few benefits, but the most significant benefit is the reduction of the cost of running DBMS on one’s own (Gelogo & Lee, 2012), (Buyya et al., 2011).

Whereby it shares information between multiple devices, and the number of these devices which expected to increase. Currently it is notable that there are a lot of companies that offer DBMS as a cloud service such as: Microsoft Azure, google, amazon EC2, GoGrid, guarantee data, Mongo lab, etc.

2.3 Query Optimization

Current relational optimizers are influenced by the techniques introduced in the system query optimizer (Patricia & Raymond, 1997; Chaudhuri & Krishnamurthy, 2006). One important contribution of this reference is a cost-based framework to obtain execution plans, which is still used with some variations in most current optimizers.
Another important contribution of (Patricia & Raymond, 1997) is a bottom-up dynamic programming, search strategy to traverse the space of candidate execution plans. This strategy needs to consider $O(N)$ expressions (Kiyoshi & Guy, 1990) for a given query. To decrease optimization time, some heuristics are used such as delaying the optimization of cartesian products, or considering only leaving-deep join trees.

The Starburst optimizer (Laura. & Christoph, 1998; Laura & Lohman, 1990) extend system-r with a more efficient, extensible approach and consists of two rule-based subsystems. In the second phase the actual execution plan is chosen.

Physical operators called LOLEPOPs can be combined in many ways to implement higher-level operators, and such combinations are expressed in a grammar production-like language (Guy & Lohman, 2001). The join enumerator in the starburst is similar to the system bottom-up enumeration scheme. The Exodus optimizer generator (Graefe & David, 1987) is the first extensible optimization framework that uses a top-down approach.

Exodus separates the optimizer's search strategy from its data model, and distinguishes between transformation rules (which map one algebraic expression into another) and implementation rules (which map an algebraic expression into an operator tree). Although it was difficult to construct efficient optimizers provide a useful foundation for the next generation of extensible optimizers.

The Volcano Optimizer Generator (William, 1993) improves the efficiency of exodus and introduces ore extensibility and effectiveness. Volcano's search algorithm combines dynamic programming with directed search based on physical properties, branch-and-bound prune and heuristic guidance. Finally, the cascades framework (Shekita & Wilms, 1993) solves some problems present in Exodus and Volcano, and improves functionality, ease of use, and robustness without compromising extensibility and efficiency.

Cascades are the state-of-the-art rule based optimization framework used in current optimizers such as Tandem's Nonstop SQL (Pedro & Celis, 1996) and Microsoft SQL server (Graefe, 1996) the cascades framework differs from the starburst in its approach to enumeration, in fact, this system does not use two distinct optimization phases as Starburst does, and the application of rules is goal-driven, as opposed to the forward-chaining rule application phase in Starburst. A detailed description of the Cascades and some extensions to the original framework appear in (Yongwen, 1998; Billings, 1997).
2.3.1 Optimization of Relational Database Management System

Relational query languages provide a high-level declarative interface to access data stored in relational database systems. With a declarative language, users (or applications acting as users) write queries stating what they want, but without specifying step-by-step instructions on how to obtain such results.

In turn, the RDBMS internally determines the best way to evaluate the input query and obtains the desired result. Structured Query Language, or SQL (Jim Melton & Alan Simon, 1993) has become the most widely used relational database languages in order to answer a given SQL query. A typical RDBMS goes through a series of steps, illustrated in Figure 2.1, which shows the input query, treated as a string of characters, is parsed and transformed into an algebraic tree that represents the structure of the query.

This step performs both syntactic and semantic checks over the input query, rejecting all invalid requests. The algebraic tree is optimized and turned into a query execution plan. A query execution plan indicates not only the operations required to evaluate the input query, but also the order in which they are performed, the algorithm used to perform each step and the way in which stored data are obtained and processed (Graefe, 1993) the query execution plan is evaluated and results are passed back to the user in the form of a relational table.

Figure 2.1: Executing SQL Queries in a Relational Database System.
Modern relational query optimizers are complex pieces of code and typically represent 40 to 50 developer-years of effort (Raghu & Johannes, 2000). As stated before, the role of the optimizer in a database system is to identify an efficient execution plan to evaluate the input query indicate in Figure 2.1. To that end, optimizers usually examine a large number of possible query plans and choose the one that is expected to result in the fastest execution.

Database queries are given in declarative languages, typically SQL. The goal of query optimization is to choose the best execution strategy for a given query under the given resource constraints. While the query specifies the user intent (i.e., the desired output), it does not specify how the output should be produced. This allows for optimization decisions, and for many queries there is a wide range of possible execution strategies, which can differ greatly in their resulting performance. This renders query optimization an important step during query processing.

The role of the optimizer is to determine the lowest cost plan for executing queries. By "lowest cost plan," it means an access path to the data that takes the least amount of time. Times invoke the optimizer for structural query language (SQL) statements when more than one execution plan is possible. The optimizer chooses what it thinks is the optimum plan. This plan persists until the statement is either invalidated or dropped by the application.

2.3.2 Architecture of Query Optimizer

Several query optimization frameworks have been proposed in the literature (David, 1987; William, 1993; Patricia & Raymond, 1997; Laura & Christoph, 1998; Graefe, 1995) and most modern optimizers rely on the concepts introduced in these references.

Although implementation details vary among specific systems, virtually all optimizers share the same basic structure (Ioannidis, 1997; Surajit, 1998) as shown in Figure 2.2.
**Figure 2.2:** Sampled Architecture of the Query Optimizer in a Database System.

For each input query, the optimizer considers a multiplicity of alternative plans. For that purpose, enumeration engine navigates through the space of candidate execution plans by applying rules.

Some optimizers have a set of rules to enumerate alternative plans (Patricia & Raymond, 1997). While others implement extensible transformational rules to navigate through the search space (Laura, 1998; Graefe, 1995).

During optimization, a cost module estimates the expected consumption of resources of each discovered query plan (resources are usually the number of I/O's, but can also include CPU time, memory, communication bandwidth, or a combination of these). Finally, once all interesting execution plans are explored, the optimizer extracts the best one, which is evaluated in the execution engine shows in Figure 2.3.

The cost estimation module is then a critical component of a relational optimizer. In general, it is not possible to obtain the exact cost of a given plan without executing it (which does not make sense during optimization). Thus, the optimizer is forced to estimate the cost of any given plan without executing it. It is then fundamental for an optimizer to rely on accurate procedures to estimate costs, since optimization is only as good as its costs estimates. Cost estimation must also be efficient, since it is repeatedly invoked during the optimization process.
The basic framework for estimating costs is based on the following recursive approach described in (Surajit, 1998) as collect statistical summaries of stored data, given an operator in the execution plan and statistical summaries for each of its sub-plans, determine tow operation statistical summaries of the output and estimated cost of executing the operator. The second step can be applied iteratively to an arbitrary tree to derive the costs of each operator. The estimated cost of a plan is then obtained by combining the costs of each of its operators. In general, the number of disk I/O's needed to manage intermediate results while executing a query plan (and thus the plan's cost) is a function of the sizes of the intermediate query results.

Therefore, the cost estimation module heavily depends on cardinality estimates of sub-plans generated during optimization. The following example illustrates how sizes of intermediate results can significantly change the plan that is chosen by an optimizer.

**Example 1:** Consider the following query template, where C is a numeric parameter.

SELECT * FROM R, S
WHERE R.x = S.y and R.a < C

Figure 2.3 shows the execution plans produced by an optimizer when instantiate C with the values 20, 200, and 2000. Three instantiated queries are almost identical.

**Figure 2.3:** Query Execution Plans for Various Instances of a Template Query
The resulting query plans are considerably different. For instance, in Figure 2.3 (A), the optimizer estimates that the number of tuples in R satisfying \( R.\ a < 20 \) is very small, so it chooses to evaluate the query as follows. First, using a secondary index over \( R.a \), it retrieves the record identifiers of all tuples in R that satisfy \( R.a < 20 \). Then, using lookups against table R, it fetches the actual tuples that correspond to those record identifiers. It performs a nested-loop join between the subset of tuples of R calculated before, and table S, which is sequentially scanned.

For the case \( C = 2000 \) in Figure 2.3 hash join, the optimizer estimates that the number of tuples of R satisfying \( R.\ a < 2000 \) is rather large, and therefore chooses to scan both tables sequentially (discarding on the y the tuples from R that do not satisfy the condition \( R.\ a < 2000 \)) and then perform a hash join to obtain the result.

(In this scenario, the lookups of the previous plan would have been too numerous, and therefore, too expensive).

Figure 2.3 merge join, shows yet another execution plan that is chosen when the number of tuples of R satisfying the predicate is neither too small nor too large. In this case, table S is scanned in increasing order of S: y using a clustered index, and table R is scanned sequentially (discarding invalid tuples on the y as before) and then sorted by R: x.

A merge join is performed on the two intermediate results it is known that if cardinality estimates are accurate, overall cost estimates are typically by no more than 10 percent (Michael & Lohman, 2001). However, cardinality estimates can be off by orders of magnitude when the underlying assumptions on the data distribution are invalid. Clearly, if the optimizer does not have accurate cardinality estimations during optimization, the \"wrong\" execution plan might be chosen for a given query.

In the previous example, if the number of tuples satisfying \( R.\ a < 2000 \) is underestimated, the optimizer could choose the less efficient plan (for that scenario) of Figure 2.3 merge join, and therefore waste time by sorting a large intermediate subset of R. In the context of adaptive query processing (Joseph & Franklin, 2003) where initial bad choices during optimization can be later corrected during query execution, accurate cardinality estimates allow the optimizer to start with a higher quality execution plan, thus minimizing the probability of dynamic changes during query execution. Henceforth for this reason, it is crucial to provide the optimizer with accurate procedures to estimate cardinality values during optimization.
Next section give an overview of statistical structures that can be used to estimate the cardinality of intermediate results generated by query sub-plans during optimization and there are a few type of join methods in DBMS.

### 2.4 Join Methods

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol $\theta$, $R_1 \bowtie R_2$, $R_1$ and $R_2$ are relations having attributes $(A_1, A_2 \ldots A_n)$ and $(B_1, B_2 \ldots B_n)$ such that the attributes do not have anything in common, that is $R_1 \cap R_2 = \emptyset$. The optimizer can be selected from multiple join methods. When the rows from two tables are joined, one table is designated the outer table and the other the inner table.

The optimizer decides which of the tables should be the outer table and which should be the inner table. During a join, the optimizer scans the rows in the outer and inner tables to locate the rows that match the join condition. The optimizer analyses the statistics for each table for example; might identify the smallest table or the table with the best selectivity for the query as outer table. If indexes exist for one or more of the tables to be joined, the optimizer takes them into account when selecting the outer and inner tables. If more than two tables are to be joined, the optimizer analyses the various combinations of joins on table pairs to determine which pair to join first, which table to join with the result of the join, and so on for the optimum sequence of joins.

The cost of a join is largely influenced by the method in which the inner and outer tables are accessed to locate the rows that match the join condition. The optimizer selects from two join methods when determining the query optimizer plan.

The current join methods as natural join, outer join are not sufficient to merge tables of database therefore necessary to find new and efficient way to improve and optimize query and RDBMS performance.

#### 2.4.1 Natural Join ($\bowtie$)

Natural join does not use any comparison operator. It does not concatenate the way a cartesian product does can perform a natural join only if there is at least one common
attribute that exists between two relations. In addition, the attributes must have the same name and domain. Natural join acts on those matching attributes where the values of attributes in both the relations are the same.

**Example Two:**

```sql
SELECT Enroll, StuId, lastName, firstName
FROM Student, Enroll
WHERE class No = 'ART 103A'
AND Enroll. StuId = Student. StuId
```

<table>
<thead>
<tr>
<th>StuId</th>
<th>last Name</th>
<th>First Name</th>
<th>Major</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1001</td>
<td>Smith</td>
<td>Tom</td>
<td>History</td>
<td>90</td>
</tr>
<tr>
<td>S1002</td>
<td>Chin</td>
<td>Ann</td>
<td>Math</td>
<td>36</td>
</tr>
<tr>
<td>S1005</td>
<td>Lee</td>
<td>Perry</td>
<td>History</td>
<td>3</td>
</tr>
<tr>
<td>S1010</td>
<td>Burns</td>
<td>Edward</td>
<td>Art</td>
<td>63</td>
</tr>
<tr>
<td>S1013</td>
<td>McCarthy</td>
<td>Owen</td>
<td>Math</td>
<td>0</td>
</tr>
<tr>
<td>S1015</td>
<td>Jones</td>
<td>Mary</td>
<td>Math</td>
<td>42</td>
</tr>
<tr>
<td>S1020</td>
<td>Rivera</td>
<td>Jane</td>
<td>CSC</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StuId</th>
<th>Class Number</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1001</td>
<td>ART103A</td>
<td>A</td>
</tr>
<tr>
<td>S1001</td>
<td>HST205A</td>
<td>C</td>
</tr>
<tr>
<td>S1002</td>
<td>ART103A</td>
<td>D</td>
</tr>
<tr>
<td>S1002</td>
<td>CSC201A</td>
<td>F</td>
</tr>
<tr>
<td>S1002</td>
<td>MTH103C</td>
<td>B</td>
</tr>
<tr>
<td>S1010</td>
<td>ART103A</td>
<td></td>
</tr>
<tr>
<td>S1010</td>
<td>MTH103C</td>
<td></td>
</tr>
<tr>
<td>S1020</td>
<td>CSC201A</td>
<td>B</td>
</tr>
<tr>
<td>S1020</td>
<td>MTH101B</td>
<td>A</td>
</tr>
</tbody>
</table>

In the example two required the use of two tables shown in the two Tables 2.1 and 2.2 and join those records into a new table shown in Table 2.3 and join those record into new table. From this table, the result show the last name and first name, this is similar to the join operation in relational algebra. SQL allows the user to do a natural join. The result of join Enroll and Student that show in Table 2.3.
2.4.2 Outer Joint

Previously, discussed at nature join, where the selects rows for the common to the participating tables to a join. What about the cases are interested in selecting elements in a table regardless of whether they are present in the second table will now need to use the SQL OUTER JOIN command. The syntax for performing an outer join in SQL is database dependent. For example, in Oracle, will place an "(+)") in the WHERE clause on the other hand of the table for which it wanted to include all the rows. Let is assume they have the following two Tables 2.4 and 2.5.

Student OUTER-EQUIJOIN Faculty

Compare Student. LastName with Faculty.name

The result of outer join query show in the Table 2.6.

<table>
<thead>
<tr>
<th>StuId</th>
<th>LastName</th>
<th>FirstName</th>
<th>Major</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1001</td>
<td>Smith</td>
<td>Tom</td>
<td>History</td>
<td>90</td>
</tr>
<tr>
<td>S1002</td>
<td>Chin</td>
<td>Ann</td>
<td>Math</td>
<td>36</td>
</tr>
<tr>
<td>S1005</td>
<td>Lee</td>
<td>Perry</td>
<td>History</td>
<td>3</td>
</tr>
<tr>
<td>S1010</td>
<td>Burns</td>
<td>Edward</td>
<td>Art</td>
<td>63</td>
</tr>
<tr>
<td>S1013</td>
<td>McCarthy</td>
<td>Owen</td>
<td>Math</td>
<td>0</td>
</tr>
<tr>
<td>S1015</td>
<td>Jones</td>
<td>Mary</td>
<td>Math</td>
<td>42</td>
</tr>
<tr>
<td>S1020</td>
<td>Rivera</td>
<td>Jane</td>
<td>CSC</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 2.4: Student Table

<table>
<thead>
<tr>
<th>FacId</th>
<th>Name</th>
<th>Department</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>F101</td>
<td>Adams</td>
<td>Art</td>
<td>Prof</td>
</tr>
<tr>
<td>F105</td>
<td>Tanaka</td>
<td>CSC</td>
<td>Instr</td>
</tr>
<tr>
<td>F110</td>
<td>Byrne</td>
<td>Math</td>
<td>Assi</td>
</tr>
<tr>
<td>F115</td>
<td>Smith</td>
<td>History</td>
<td>Asso</td>
</tr>
<tr>
<td>F221</td>
<td>Smith</td>
<td>CSC</td>
<td>Prof</td>
</tr>
</tbody>
</table>

Table 2.5: Faculty Table

Table 2.6: Outer Join for Student and Faculty Tables
The outer equijoin uses to search full tables left and right in current example search about student last name in Table 2.4 to compare last name in student Table 2.4 with name in the faculty in the Table 2.5 to finding similar name then the result will put the required record in the result Table 2.6 otherwise leave the record of right table is null.

2.4.3 Left Outer Joint

In a left outer join, all rows from the first table mentioned in the SQL query is selected, regardless whether there is a matching row on the second table mentioned in the SQL query. Let is assume having the following two tables.

**Table 2.7: Student Table**

<table>
<thead>
<tr>
<th>StudId</th>
<th>LastName</th>
<th>FirstName</th>
<th>Major</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1001</td>
<td>Smith</td>
<td>Tom</td>
<td>History</td>
<td>90</td>
</tr>
<tr>
<td>S1002</td>
<td>Chin</td>
<td>Ann</td>
<td>Math</td>
<td>36</td>
</tr>
<tr>
<td>S1005</td>
<td>Lee</td>
<td>Perry</td>
<td>History</td>
<td>3</td>
</tr>
<tr>
<td>S1010</td>
<td>Burns</td>
<td>Edward</td>
<td>Art</td>
<td>63</td>
</tr>
<tr>
<td>S1013</td>
<td>McCarthy</td>
<td>Owen</td>
<td>Math</td>
<td>0</td>
</tr>
<tr>
<td>S1015</td>
<td>Jones</td>
<td>Mary</td>
<td>Math</td>
<td>42</td>
</tr>
<tr>
<td>S1020</td>
<td>Rivera</td>
<td>Jane</td>
<td>CSC</td>
<td>15</td>
</tr>
</tbody>
</table>

**Table 2.8: Faculty Tables**

<table>
<thead>
<tr>
<th>FacId</th>
<th>Name</th>
<th>Department</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>F101</td>
<td>Adams</td>
<td>Art</td>
<td>Professor</td>
</tr>
<tr>
<td>F105</td>
<td>Tanaka</td>
<td>CSC</td>
<td>Instructor</td>
</tr>
<tr>
<td>F110</td>
<td>Byrne</td>
<td>Math</td>
<td>Assistant</td>
</tr>
<tr>
<td>F115</td>
<td>Smith</td>
<td>History</td>
<td>Associate</td>
</tr>
<tr>
<td>F221</td>
<td>Smith</td>
<td>CSC</td>
<td>Professor</td>
</tr>
</tbody>
</table>

**Table 2.9: Left Join for Student and Faculty Tables**

<table>
<thead>
<tr>
<th>StudId</th>
<th>LastName</th>
<th>FirstName</th>
<th>Major</th>
<th>Credits</th>
<th>FacId</th>
<th>Name</th>
<th>Department</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1001</td>
<td>Smith</td>
<td>Tom</td>
<td>History</td>
<td>90</td>
<td>F221</td>
<td>Smith</td>
<td>CSC</td>
<td>Professor</td>
</tr>
<tr>
<td>S1001</td>
<td>Smith</td>
<td>Tom</td>
<td>History</td>
<td>90</td>
<td>F115</td>
<td>Smith</td>
<td>History</td>
<td>Associate</td>
</tr>
<tr>
<td>S1002</td>
<td>Chin</td>
<td>Ann</td>
<td>Math</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1005</td>
<td>Lee</td>
<td>Perry</td>
<td>History</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1010</td>
<td>Burns</td>
<td>Edward</td>
<td>Art</td>
<td>63</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1013</td>
<td>McCarthy</td>
<td>Owen</td>
<td>Math</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1015</td>
<td>Jones</td>
<td>Mary</td>
<td>Math</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1020</td>
<td>Rivera</td>
<td>Jane</td>
<td>CSC</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In the left join, all rows in the left table to kept in the result and compare the last name in student Table 2.7 with name in the faculty Table 2.8 for the column name in the faculty, if the same name have found in the name of faculty table, in this case save all record of faculty table in the result Table 2.9 otherwise the result will be null.

### 2.4.4 Right-Outer Join

The right outer join keyword returns all rows from the right table with the matching rows in the left table. The result is NULL in the left side when there is no match.

**Student RIGHT-OUTER-EQUIJOIN**

**Table 2.10: Student Table**

<table>
<thead>
<tr>
<th>StudId</th>
<th>LastName</th>
<th>FirstName</th>
<th>Major</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1001</td>
<td>Smith</td>
<td>Tom</td>
<td>History</td>
<td>90</td>
</tr>
<tr>
<td>S1002</td>
<td>Chin</td>
<td>Ann</td>
<td>Math</td>
<td>36</td>
</tr>
<tr>
<td>S1005</td>
<td>Lee</td>
<td>Perry</td>
<td>History</td>
<td>3</td>
</tr>
<tr>
<td>S1010</td>
<td>Burns</td>
<td>Edward</td>
<td>Art</td>
<td>63</td>
</tr>
<tr>
<td>S1013</td>
<td>McCarthy</td>
<td>Owen</td>
<td>Math</td>
<td>0</td>
</tr>
<tr>
<td>S1015</td>
<td>Jones</td>
<td>Mary</td>
<td>Math</td>
<td>42</td>
</tr>
<tr>
<td>S1020</td>
<td>Rivera</td>
<td>Jane</td>
<td>CSC</td>
<td>15</td>
</tr>
</tbody>
</table>

**Table 2.11: Faculty**

<table>
<thead>
<tr>
<th>FacId</th>
<th>Name</th>
<th>Department</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>F101</td>
<td>Adams</td>
<td>Art</td>
<td>Professor</td>
</tr>
<tr>
<td>F105</td>
<td>Tanaka</td>
<td>CSC</td>
<td>Instructor</td>
</tr>
<tr>
<td>F110</td>
<td>Byrne</td>
<td>Math</td>
<td>Assistant</td>
</tr>
<tr>
<td>F115</td>
<td>Smith</td>
<td>History</td>
<td>Associate</td>
</tr>
<tr>
<td>F221</td>
<td>Smith</td>
<td>CSC</td>
<td>Professor</td>
</tr>
</tbody>
</table>

**Table 2.12: Result of Right Outer join**

<table>
<thead>
<tr>
<th>StudId</th>
<th>LastName</th>
<th>FirstName</th>
<th>Major</th>
<th>Credits</th>
<th>FacId</th>
<th>Name</th>
<th>Department</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1001</td>
<td>Smith</td>
<td>Tom</td>
<td>History</td>
<td>90</td>
<td>F221</td>
<td>Smith</td>
<td>CSC</td>
<td>Professor</td>
</tr>
<tr>
<td>S1001</td>
<td>Smith</td>
<td>Tom</td>
<td>History</td>
<td>90</td>
<td>F115</td>
<td>Smith</td>
<td>History</td>
<td>Associate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F101</td>
<td>Adams</td>
<td>Art</td>
<td>Professor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F105</td>
<td>Tanaka</td>
<td>CSC</td>
<td>Instructor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F110</td>
<td>Byrne</td>
<td>Math</td>
<td>Assistant</td>
</tr>
</tbody>
</table>
2.5 Multi Join Query Optimization in Relational Database Management System

Query optimization is a function of many relational database management systems. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans. Multi-join query is one of the basic operations while using database. Therefore, Multi-join query optimization is of great necessity to improve database performance.

There are often other cost metrics in addition to execution time that are relevant to compare query plans (Trummer & Immanuel, 2015). In a cloud computing scenario for instance, one should compare query plans not only in terms of how much time they take to execute but also in terms of how much money spending their execution costs. The context of approximate query optimization, it is possible to execute query plans on randomly selected samples of the input data in order to obtain approximate results with reduced execution overhead.

For example, in a database system enhanced with inference capabilities, a simple query involving a rule with multiple definitions may expand to more than one actual query that has to be run over the database.

In the past few years, several attempts have been made to extend the benefits of the database approach in business to other areas, such as artificial intelligence and engineering design automation. Traditionally, query optimizers like (Chaudhuri, 2006) optimize queries one at a time and do not identify any commonalities in queries, resulting in repeated computations. As observed in (Rosenthal & Chakravarthy, 1988; Sellis, 1988) exploiting common results can lead to significant performance gains.

This is known as multi-query optimization. Existing techniques for multi-query optimization assume that all intermediate results are materialized (Cosar & Srivastava, 2008; Roy & Seshadri, 2000; Deshpande et al., 1998).

They assume that if a common subexpression is to be shared, it will be materialized and read whenever it is required subsequently. Current multi-query optimization techniques do not try to exploit pipelining of results to all the users of the common subexpression.


Bramley & Chiu (2000). A component based services architecture for building distributed applications in processing of HPDC.


Chaudhuri & krishnamurthy (2006) .optimization queries with matalrized views . proc.11 the ICDE,190,200.


Christopher Beer & Tim Hendtlass (2012). Improving Exploration in Ant Colony Optimisation with Antennation, IEEE must be obtained for all other uses.


Guido M. (2006). Analysis of two existing and one new dynamic programming, VLDB ’06 Proceedings of the 32nd international conference on Very large data, Pages 930-941


The Social Physiology of Honey Bee Colonies.


Shekita Lapis, G., & wilim (1993). Starburst midnight: As the dust clears. IEEE Transactions on Knowledge and Data Engineering, 2(1).


Thusoo A & Borthakur D. (2010). Data Warehousing and Analytics Infrastructure at Facebook. Data warehouse scalability, data discovery, resource sharing, distributed file system, Hadoop, Hive, Facebook, Scribe, log aggregation, analytics, mapreduce, distributed systems.


