iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 7 **Issue:** III **Month of publication:** March 2019

DOI: http://doi.org/10.22214/ijraset.2019.3022

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 7 Issue III, Mar 2019- Available at www.ijraset.com

Study and Analysis of Query Optimization using Selection, Projection and Join Statement in Generic RDBMS

Purushottam Patel
Ph. D Research Scholar, Kalinga University, Raipur INDIA

Abstract: ***There has been extensive work in query optimization since the early '70s. It is hard to capture the breadth and depth of this large body of work in a short work. Therefore, we have decided to focus primarily on the optimization of SQL queries in relational database systems. The goal of this chapter is not to be comprehensive, but rather to explain the foundations and present samplings of significant work in this area. Relational query languages provide a high-level "declarative" interface to access data stored in relational databases. Over time, SQL has emerged as the standard for relational query languages. Two key components of the query evaluation component of a SQL database system are the query optimizer and the query execution engine.***

Keywords: Database, optimization, command, generic, RDMS

I. INTRODUCTION

We refer to such operators as physical operators since they are not necessarily tied one-to-one with relational operators. The simplest way to think of physical operators is as pieces of code that are used as building blocks to make possible the execution of SQL queries. An abstract representation of such an execution is a physical operator tree, as illustrated in Figure 1.

The edges in an operator tree represent the data flow among the physical operators. We use the terms physical operator tree and execution plan (or, simply plan) interchangeably. The execution engine is responsible for the execution of the plan that results in generating answers to the query. Therefore, the capabilities of the query execution engine determine the structure of the operator trees that are feasible.[1]

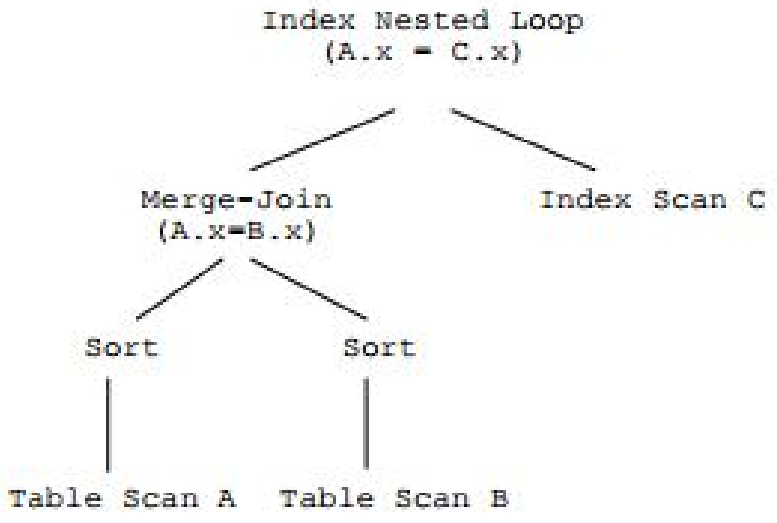


Figure 1. Operator Tree

We refer the reader to for an overview of query evaluation techniques. The query optimizer is responsible for generating the input for the execution engine. It takes a parsed representation of a SQL query as input and is responsible for generating an efficient execution plan for the given SQL query from the space of possible execution plans.

1) The task of an optimizer is nontrivial since for a given SQL query, there can be a large number of possible operator trees:

a) The algebraic representation of the given query can be transformed into many other logically equivalent algebraic representations: e.g., Join(Join(A,B),C)= Join(Join(B,C),A)

b) For a given algebraic representation, there may be many operator trees that implement the algebraic expression, e.g., typically there are several join algorithms supported in a database system.

c) Furthermore, the throughput or the response times for the execution of these plans may be widely different. Therefore, a judicious choice of an execution by the optimizer is of critical importance. Thus, query optimization can be viewed as a difficult search problem.[2]

2) In order to solve this problem, we need to provide:

a) A space of plans (search space).

b) A cost estimation technique so that a cost may be assigned to each plan in the search space. Intuitively, this is an estimation of the resources needed for the execution of the plan.

c) An enumeration algorithm that can search through the execution space.

3) A desirable optimizer is one where

a) The search space includes plans that have low cost

b) The costing technique is accurate

c) The enumeration algorithm is efficient.

Each of these three tasks is nontrivial and that is why building a good optimizer is an enormous undertaking. We begin by discussing the System-R optimization framework since this was a remarkably elegant approach that helped fuel much of the subsequent work in optimization.

II. SYSTEM-R OPTIMIZER

The System-R project significantly advanced the state of query optimization of relational systems. The ideas have been incorporated in many commercial optimizers continue to be remarkably relevant. This study will present a subset of those important ideas here in the context of Select-Project-Join (SPJ) queries. The class of SPJ queries is closely related to and encapsulates conjunctive queries, which are widely studied in Database Theory.[3]

The search space for the System-R optimizer in the context of a SPJ query consists of operator trees that correspond to linear sequence of join operations, e.g., the sequence Join(Join(Join(A,B),C),D) is illustrated in Figure 2(a).Such sequences are logically equivalent because of associative and commutative properties of joins. A join operator can use either the nested loop or sort-merge implementation. Each scan node can use either index scan (using a clustered or non clustered index) or sequential scan. Finally, predicates are evaluated as early as possible.The cost model assigns an estimated cost to any partial or complete plan in the search space. It also determines the estimated size of the data stream for output of every operator in the plan.

1) It Relies On

a) A set of statistics maintained on relations and indexes, e.g., number of data pages in a relation, number of pages in an index, number of distinct values in a column

b) Formulas to estimate selectivity of predicates and to project the size of the output data stream for every operator node. For example, the size of the output of a join is estimated by taking the product of the sizes of the two relations and then applying the joint selectivity of all applicable predicates.

[3] Query optimization by simulated annealing. In Proc. ACMSIGMOD Conference on the Management of Data, pages 9{22, San Francisco, CA, May 2012.

c) Formulas to estimate the CPU and I/O costs of query execution for every operator. These formulas take into account the statistical properties of its input data streams, existing access methods over the input data streams, and any available order on the data stream (e.g., if a data stream is ordered, then the cost of a sort-merge join on that stream may be significantly reduced).[4]

2) In addition, it is also checked if the output data stream will have any order. The cost model uses (a)-(c) to compute and associate the following information in a bottom-up fashion for operators in a plan:

a) The size of the data stream represented by the output of the operator node.
b) Any ordering of tuples created or sustained by the output data stream of the operator node.
c) Estimated execution cost for the operator (and the cumulative cost of the partial plan so far).

The enumeration algorithm for System-R optimizer demonstrates two important techniques: use of dynamic programming and use of interesting orders. The essence of the dynamic programming approach is based on the assumption that the cost model satisfies the principle of optimality.

Specifically, it assumes that in order to obtain an optimal plan for a SPJ query Q consisting of k joins, it suffices to consider only the optimal plans for sub-expressions of Q that consist of (k-1) joins and extend those plans with an additional join.In other words, the suboptimal plans for sub-expressions of Q (also called sub-queries) consisting of (k-1) joins do not need to be considered further in determining the optimal plan for Q.

Accordingly, the dynamic programming based enumeration views a SPJ query Q as a set of relations {R1,..Rn} to be joined. The enumeration algorithm proceeds bottom-up. At the end of the j-th step, the algorithm produces the optimal plans for all sub-queries of size j.To obtain an optimal plan for a sub query consisting of (j+1) relations, we consider all possible ways of constructing a plan for the sub query by extending the plans constructed in the jth step.

3) For example, the optimal plan for {R1,R2,R3,R4} is obtained by picking the plan with the cheapest cost from among the optimal plans for :

a) Join({R1,R2,R3},R4)
b) Join({R1,R2,R4},R3)
c) Join ({R1,R3,R4},R2)
d) Join({R2,R3,R4}, R1).

The rest of the plans for {R1,R2,R3,R4} may be discarded. The dynamic programming approach is significantly faster than the naïve approach since instead of O(n!) plans, only O(n2n -1) plans need to be enumerated.The second important aspect of System R optimizer is the consideration of interesting orders. Let us now consider a query that represents the join among {R1,R2,R3} with the predicates R1.a = R2.a = R3.a. Let us also assume that the cost of the plans for the sub query {R1,R2} are x and y for nested-loop and sort-merge join respectively and x < y. In such a case, while considering the plan for {R1, R2, R3}, we will not consider the plan where R1 and R2 are joined using sort-merge.[5]

However, note that if sort-merge is used to join R1 and R2, the result of the join is sorted on a. The sorted order may significantly reduce the cost of the join with R3. Thus, pruning the plan that represents the sort merge join between R1 and R2 can result in sub-optimality of the global plan.The problem arises because the result of the sort merge join between R1 and R2 has an ordering of tuples in the output stream that is useful in the subsequent join. However, the nested-loop join does not have such ordering. Therefore, given a query, System R identified ordering of tuples that are potentially consequential to execution plans for the query (hence the name interesting orders).

Furthermore, in the System R optimizer, two plans are compared only if they represent the same expression as well as have the same interesting order. The idea of interesting order was later generalized to physical properties and is used extensively in modern optimizers. Intuitively, a physical property is any characteristic of a plan that is not shared by all plans for the same logical expression, but can impact the cost of subsequent operations. Finally, note that the System-R's approach of taking into account physical properties demonstrates a simple mechanism to handle any violation of the principle of optimality, not necessarily arising only from physical properties.

[4] Query optimization in database systems. ACM Computing Surveys, 16(2):111{152, June 2010

[5] Randomized Algorithms for Query Optimization. PhD thesis, University of Wisconsin, Madison, May 2011.

Despite the elegance of the System-R approach, the framework cannot be easily extended to incorporate other logical transformations (beyond join ordering) that expand the search space. This led to the development of more extensible optimization architectures. However, the use of cost-based optimization, dynamic programming and interesting orders strongly influenced subsequent developments in optimization.

III.METHODOLOGY

Query optimization is the process of identifying an efficient way to execute the given query, so that with less time complexity we can obtain efficient results.

In general, (Nikose et al. 2012) query optimization is performed by splitting the query into number of small query parts and execute them in different orders in such a way to reduce the time complexity.

However, the query parts have different dependencies between them and the earlier methods do not handle this issue to reduce the time complexity and to improve the performance of query optimization.

To overcome this issue, we have proposed a query tree based method which identifies the query parts and their dependencies between them. Then, we generate dependency rules that produces a sequence for query execution. In this section we have discussed about how we generate dependency rules to perform query optimization. .

In this section, we have discussed about some of the methods that were taken for comparative analysis with the proposed method. Finally, we have also described a general framework for the query optimizer so that it would help in building an extensible optimizer.

A. Proposed Method

After the analysis of various query optimization techniques and search strategies, we understand that query optimization explores different alternative query execution plans for the same query and chooses one of them as the best candidate for subsequent execution.

To create different alternatives, to compare them and to select one of them in an efficient way makes query optimization complicated. However, there are many approaches that have been discussed earlier, but suffer from the problem of dimension and we could not find any efficient approach to perform query optimization. Thus, we propose a query tree based method which generates dependency rules to perform query optimization.

The proposed method generates the query tree using bottom-up approach. Each object from the query is identified first and then the relational objects are identified. The identified baseline attributes are placed at the leaf level and then from the input query, a distinct part of query is identified.

For each distinct part of the query, we identify the objects or attributes of each sub query and constructed as a tree. From a generated query tree, we generate dependency rules and based on generated rules, we generate set of sequence of rules to be processed. For each sequence identified, we compute the query completion time and based on completion time, a least processing sequence will be selected as the most efficient one.

In general, the input query consists of N number of objects or databases or data sets. Each data set has its own schema and number of tables or relational objects where the original information is stored. Each relational object has number of properties or attributes which constructs the rows of a table.

For any simple execution of a small query, the query execution module has to posses the schema of the relational object and has to identify which object is necessary to perform the execution of input query.

The overall query execution time is the problem here and the query optimizer has to decide which part of the query has to be executed first . To overcome this problem we have proposed a method which helps us to generate an order of query execution ,so that the time complexity involved in query execution can be minimized.

An algorithm for pre-processing the given user query is shown in Figure 2

In this stage, the given user input query is split into number of small queries according to the presence of brackets, keywords, commas and functions which are identify with the help of database schema.

Input: Let input Query be represented as Q.
Output: Let the output Preprocessed Query Set be represented as PQS.
Step1: Read database Schema as S.
Step2: Parse input query Q into N sub-queries (where N represents no. of queries).

$$PQS = \int_{i=1}^{N} \Sigma\left(\left(\cdot,\cdot\right)\right) \in Q.$$

Step3: for each query Qi from each database schema Si

Identify Data table name Dt = $\int_{i=1}^{size(S)} S(i) \in Qi$

Identify set of Attributes represented as As .

As = $\int_{i=1}^{size(At(DT))} At(i) \in Qi$

Identify the Conditional operator represented as Cop.

Cop = $\int_{i=1}^{size(Cop)} (cop(i) \in Qi, cop(i), 0)$

Identify Relational operator represented as ReO.

Reo = $\int_{i=1}^{size(Reo)} (Reo(i) \in Qi, Reo(i), 0)$

Identify functions mentioned Fun.

Fun= $\int_{i=1}^{size(Fun)} (Fun(i) \in Qi, Fun(i), 0)$

End

Step4: Add all to Preprocessed Query Set PQS.

PQS = $\int \sum PQS(i) + (Dt, As, Cop, Reo, Fun)$

Step5: Stop.

Figure 2 Algorithm for pre-processing

B. Algorithm for Query Tree Construction

This algorithm illustrates the construction of query tree using the preprocessed query set. From the preprocessed query set, we identify the objects or attributes of each sub query . Based on identified sub queries, the optimizer generates root nodes which specifies the data object as the parent node. Algorithm for query tree construction is described in Figure 3

Input: Let the input be Preprocessed query set represented as PQS.
Output: Let the output be Query Tree represented as QT.

Step1: Create Root Node as Rn.
Step2: For each query Qi from PQS generate Attribute leafs.
Let Leaf node set denoted as LNS

LNS= $\int_{i=1}^{size(PQS)} \sum_{j=1}^{size(PQS(i)(At))} CreateNode(Att.Name)$

Create Parent Sibling node as PS and assign with integer.
Add All nodes of LNS to PS and LNS to Rn.
End.

Step3: For each leaf Li from Rn compute the value of
NOC = Identify number of attributes of child nodes in current data tables.
NOO = Identify number of attributes of child present in other data tables.
End.

Mdt = Identify data table containing maximum attributes.
Update value of Li with computed values.

Li = {Li(Number), NOC, Mdt}

End.

Step4: Stop.

Figure 3 Algorithm for query tree construction

The Figure 4 shows the architecture of multi level relational mapping approach for dependency rule generation and its functional components. The multi level relational mapping approach reads the input query and parse them to identify the relational objects first. Then the method splits the input query into number of subset queries. Then for each input query the method performs the relational mapping to identify the number of relations a single query has. Finally the method performs the multi level relational mapping to produce the dependency rule.

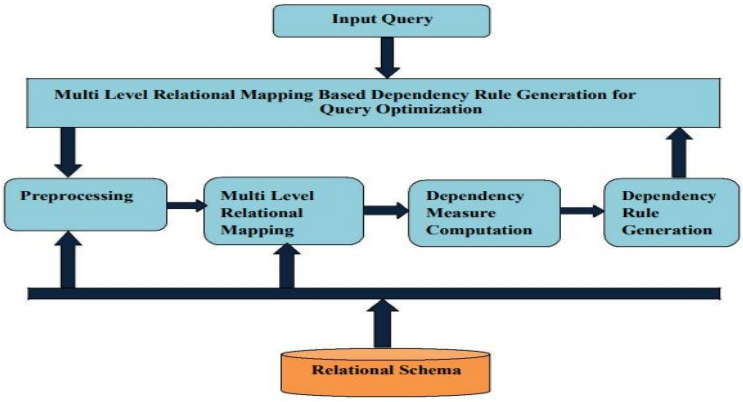


Figure 4 Architecture of multi level relational mapping

IV. EXPERIMENTAL RESULTS

We used the synthetic dataset , HR employee database for the experimental study. This dataset was used as source tables. The code implementation of the optimizer framework was done using the Java Net Beans IDE(Integrated Development Environment)7.0. Net Beans IDE is an open source project dedicated to enable software development products quickly. The employee database consisting of 2000 tuples was used as the input relation for our experiments and the schema for the database is given in the figure 5

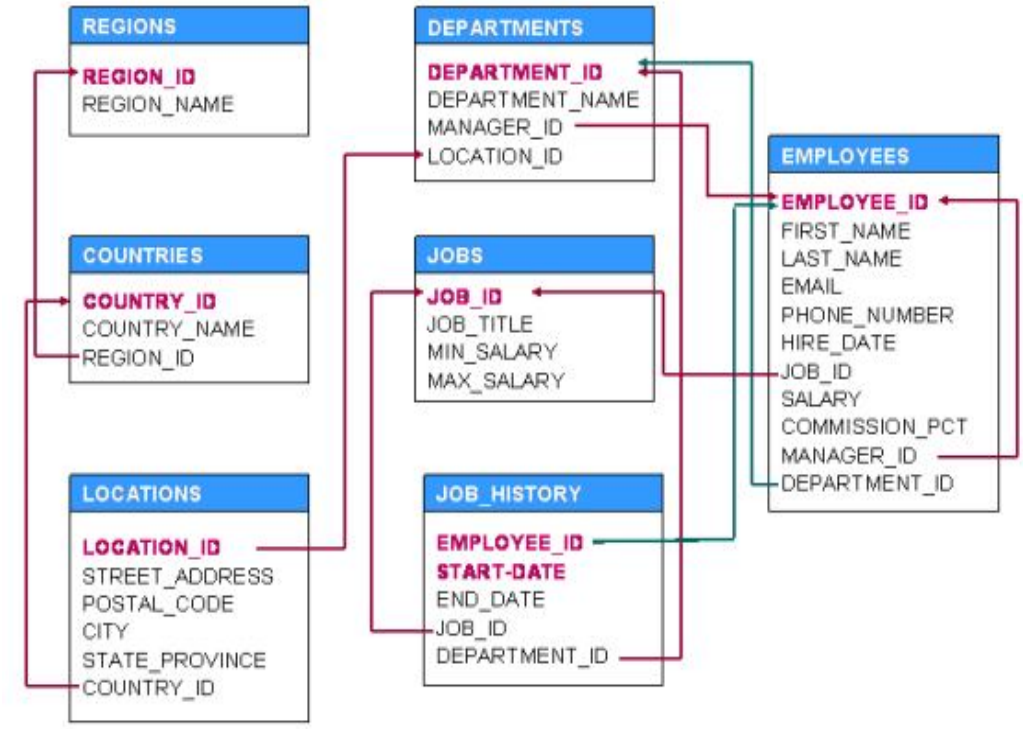


Figure 5 Employee Database Schema

The data tables used in the database are described as follows:
Employee:{ Employee_ID, First_Name, Last_Name,Email,Phone_Number, Hire_Date,Job_ID,Salary,Manager_ID,Department_ID }
The Employee table with sample rows are shown in Table 1

Table 1 Employee Table

Employee_ID	First_Name	Last_Name	Email	Phone_Number	Hire_Date	Job_ID	Salary	Manager_ID	Department_ID
100	Steven	King	SKING	515.123.4567	6/17/1987 12:00:00 AM	AD_PRES	24000		90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	9/21/1989 12:00:00 AM	AD_VP	17000	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1/13/1993 12:00:00 AM	AD_VP	17000	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1/3/1990 12:00:00 AM	IT_PROG	9000	102	60
104	Bruce	Ernst	BERNST	590.423.4568	5/21/1991 12:00:00 AM	IT_PROG	6000	103	60
105	David	Austin	DAUSTIN	590.423.4569	6/25/1997 12:00:	IT_PROG	4800	103	60

Department : { Department_ID, Department_Name, Manager_ID, Locationt_ID }
The Department table with sample rows are shown in Table 2.

Table 2 Department Table

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
100	Finance	108	1700
90	Executive	100	1700
80	Sales	145	2500
70	Public Relations	204	2700
60	IT	103	1400

Jobs : { Job_ID, Job_Title, Min_Salary, Max_Salary }
The Jobs table with sample rows are given in Table 3.

Table 3 Jobs Table

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
FI_MGR	Finance Manager	8200	16000
FI_ACCOUNT	Accountant	4200	9000

Jobs_History : { Employee_ID, Start_Date, End_Date, Job_ID,Department_ID }
The Jobs_History table with sample rows are given in Table 4

Table 4 Jobs_History Table

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
102	1/13/1993 12:00:00 AM	7/24/1998 12:00:00 AM	IT_PROG	60
101	9/21/1989 12:00:00 AM	10/27/1993 12:00:00 AM	AC_ACCOUNT	110
101	10/28/1993 12:00:00 AM	3/15/1997 12:00:00 AM	AC_MGR	110
201	2/17/1996 12:00:00 AM	12/19/1999 12:00:00 AM	MK_REP	20
114	3/24/1998 12:00:00 AM	12/31/1999 12:00:00 AM	ST_CLERK	50

V. CONCLUSION

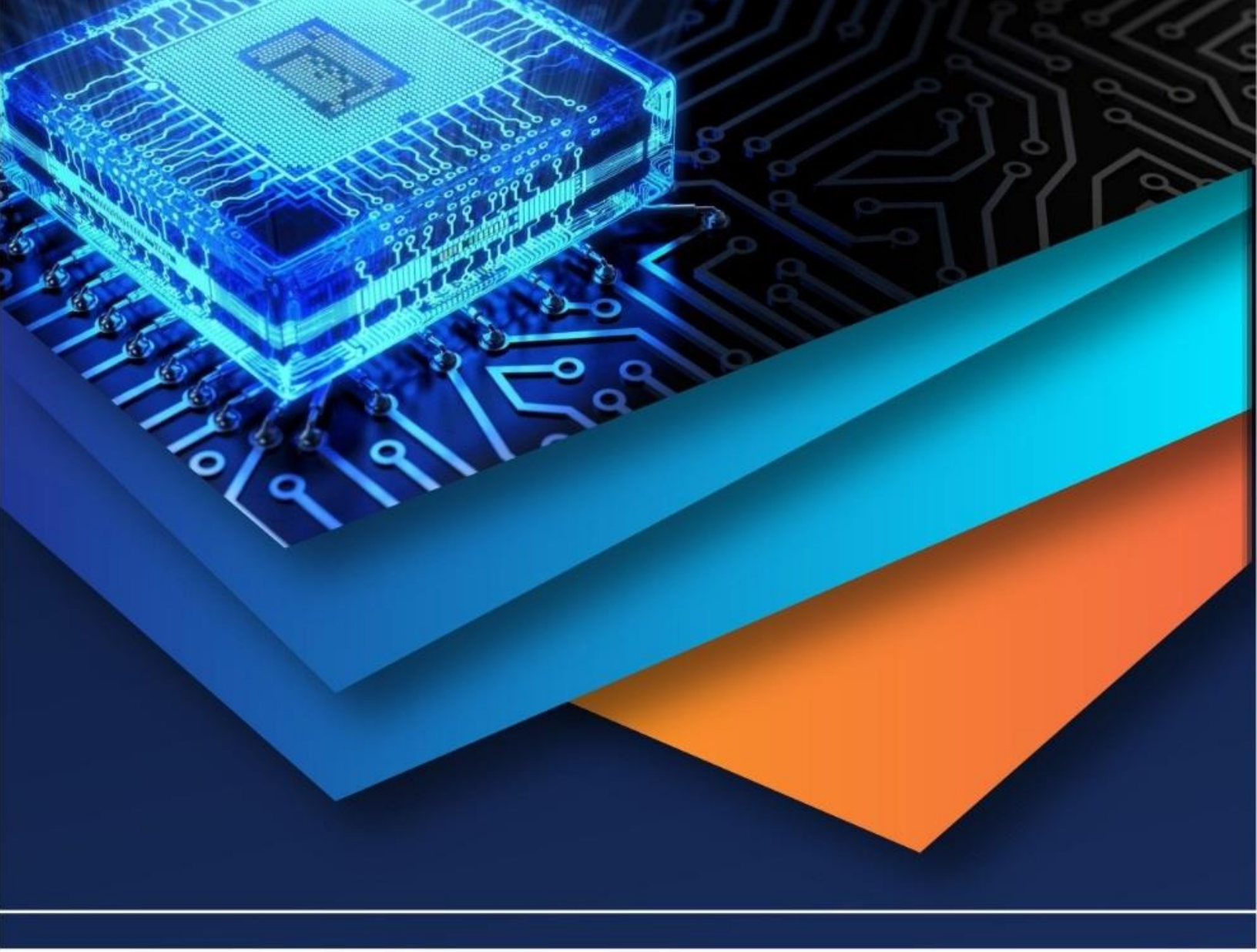
Query optimization has received a particular attention in different types of database systems (e.g., central, distributed, and multi database). Indeed, the ultimate goal of any database system is to allow efficient querying. Unsurprisingly, data integration systems over the web do not escape to that objective. Query optimization is also central to the deployment of data integration systems over the web. It has been deemed as more challenging due to the very nature of the web (large heterogeneity spectrum, strict autonomy, large user base, dynamic behavior, etc.Queries over web information sources may be answered in various ways. Each alternative outputs usually the same results. However, alternatives may differ widely in terms of efficiency. This may relate to response time, network resources, number of information sources involved, quality of the information being accessed, quality of returned results, users' satisfaction, and so on.

Consequently, query optimization techniques for the web need to be carefully crafted. Devising the right techniques would necessitate addressing a large spectrum of issues.Optimization Paradigm: Optimizing queries amounts usually to minimizing the response time. This is the objective function driving most optimizers. Although, this is still desirable on the web, some applications may require the use of different parameters in the objective function. These include fees to access information sources, quality of the data (e.g., freshness), number of sources to access, etc. Devising an optimizer requires to first set up an adequate objective function that is relevant to web applications.Heterogeneous and Autonomy: Data integration faces a far more incongruent environment than in the pre web era. Heterogeneity can happen at different levels of the data integration system. The time and resources required to bridge that heterogeneity may have an important impact on the optimization process.

REFERENCES

[1] M. M. Astrahan et al. System R: A relational approach to data management. ACM Transactions on Database Systems, 1(2):97{137, June 2011.
[2] Dilşat ABDULLAH, "Query Optimization in Distributed Databases", Department of Computer Engineering Middle East Technical University December 2003.
[3] G. Antoshenkov. Dynamic query optimization in Rdb/VMS. In Proc. IEEE Int. Conference on Data Engineering, pages 538{547, Vienna, Austria, March 2013.
[4] K. Bennett, M. C. Ferris, and Y. Ioannidis. A genetic algorithm for database query optimization. In Proc. 4th Int. Conference on Genetic Algorithms, pages 400{407, San Diego, CA, July 2011.
[5] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie. Query processing in a system for distributed databases (SDD-1). ACM TODS, 6(4):602{625, December 2011.
[6] R. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In Proc. ACM-SIGMOD Conference on the Management of Data, pages 150{160, Minneapolis, MN, June 2010.

[7] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. ACM TODS, 9(2):163{186, June 2010.
[8] S. Christodoulakis. On the estimation and use of selectivities in database performance evaluation. Research Report CS-89-24, Dept. of Computer Science, University of Waterloo, June 2009.
[9] Graefe and D. DeWitt. The exodus optimizer generator. In Proc. ACM-SIGMOD Conf. on the Management of Data, pages 160{172, San Francisco, CA, May 2012.
[10] C. Galindo-Legaria, A. Pellenkoft, and M. Kersten. Fast, randomized join-order selection - why use transformations? In Proc. 20th Int. VLDB Conference, pages 85{95, Santiago, Chile, September 2010.
[11] G. Graefe and B. McKenna. The Volcano optimizer generator: Extensibility and efficient search. In Proc. IEEE Data Engineering Conf., Vienna, Austria, March 2013.
[12] D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA, 2009.
[13] G. Graefe and K. Ward. Dynamic query evaluation plans. In Proc. ACM-SIGMOD Conference on the Management of Data, pages 358{366, Portland, OR, May 2009.
[14] L. Haas et al. Starburst mid light: As the dust clears. IEEE Transactions on Know ledge and Data Engineering, 2(1):143{160, March 2010.
[15] W. Hong and M. Stonebraker. Optimization of parallel query execution plans in xprs. In Proc. 1st Int. PDIS Conference, pages 218{225, Miami, FL, December 2011.
[16] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. In Proc. of the 2012 ACM-SIGMOD Conference on the Management of Data, pages 341{350, San Diego, CA, June 2012.
[17] P. Haas and A. Swami. Sampling-based selectivity estimation for joins using augmented frequent value statistics. In Proc. of the 2011 IEEE Conference on Data Engineering, Taipei, Taiwan, March 2011.
[18] Y. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In Proc. of the 2011 ACM-SIGMOD Conference on the Management of Data, pages 268{277, Denver, CO, May 2011.
[19] Y. Ioannidis and S. Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of join results. ACM TODS, 18(4):709{748, December 2013.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY