Analysis of Query Processing and Optimization

Nimra Memon, Muhammad Saleem Vighio, Shah Zaman Nizamani, Niaz Ahmed Memon, Adeel Riaz Memon, Umair Ramzan Shaikh

Abstract—**Modern database management systems are complex programs which get user queries, translate them in internal representation necessary for data access and provide results in best possible way; that is, results which take less execution-time and consume fewer resources. However, as the data is increasing at rapid pace, there is always a need of systems which meet organizational requirements. Better systems can only be developed if internal working of (existing) systems is understood very well. This paper presents a tool that helps in understanding how (high-level) user queries are translated in internal representation necessary for data access. Furthermore, the tool also implements few optimization techniques to produce alternative execution plans and their corresponding execution time. Paper also presents a discussion on which plans are better based on the computational analysis.**

***Index Terms*— SQL, Relational Algebra, Query Processing, Query Optimization**

I. INTRODUCTION

In our everyday life, we deal with databases in one or another way for e.g., depositing or withdrawing money using online banking, paying our electricity bills, purchasing a book online, or searching for historic place using google.com [1] [2]. Actually, this communication for intended data is made possible with the help of database management system (DBMS) software. In relational database management systems (RDBMs), user queries usually provided in Structured Query Language (SQL) are translated in internal representation (i.e., relational algebra) necessary for data access [3]. A subprogram called query parser parses user query for syntax and semantic errors, translates it into an equivalent expression in relational algebra, and decomposes it in query blocks. Relational algebraic expression is then further analyzed and optimized by a query optimizer (a subprogram in DBMS package) that generates equivalent query plans and chooses a best one in terms of execution time and consumption of resources.

Database management systems are complex programs responsible for performing all this query processing and optimization. The pace at which data recording needs are increasing one can confidently say that future systems will be even more complex as they have to deal with voluminous amounts of data. Therefore, for meeting future requirements it is very important that existing working of DBMS software must be understood very well. In this paper, we present a tool that helps in understanding how user queries provided in structured query language (SQL) are translated in internal representation (relational algebra). Furthermore, for each input query the tool implements three optimization strategies (simple, elimination of Cartesian product, push selection) to produce alternative execution plans and their corresponding execution time. Based on the produced results, paper presents discussion on which optimization strategies are better.

Nimra Memon, Muhammad Saleem Vighio, Shah Zaman Nizamani, Adeel Riaz Memon, Umair Ramzan Shaikh, Information Technology Department, Quaid-e-Awam University of Engineering, Science & Technology Nawabshah, Pakistan. Niaz Ahmed Memon, Adl. Plant Manager (Maint:)-III, 600MW Combined Cycle Power Station (CCPS) Thermal P ower Station (TPS), Genco-II Guddu. Email: nimramemon88@hotmail.com

II. LITERATURE REVIEW

Much research has been carried out on query translation and optimization for e.g., in [3] a detailed discussion on query processing is provided. Similar research has also been provided in [4] and [5] which have remained very useful for the development of the part of our tool which translates user queries from SQL to relational algebra. Similar to our work, Bendre M. *et. al.,* in [6] have developed a tool that translates relational algebraic expressions to relational calculus (SQL). Our inspiration came also from the work presented in [6]. However, as opposed to the translation performed in [6], our tool performs forward translation i.e., from relational calculus (SQL) to relational algebra which is the case with almost all types of database management systems developed so far. Additionally, our tool provides connectivity to the databases created in SQL server and also provides actual query results. The parser in our tool generates syntactic errors if SQL queries are not well-formed i.e., either the queries do not follow the allowed syntax or refer to tables/database which do not exist. The tool in [6] lacks these key features. The work provided in [4] has also remained useful for the development of our tool. In [4] a tool supported translation is provided from a relevant subset of SQL into relational algebra. The translation is syntax-directed, with translation rules associated with grammar productions; each production corresponds to a particular type of SQL sub-query. Translator defines the semantics of the SQL language, and can be used for the proof of equivalence of SQL queries which are syntactically different.

For the optimization part of our tool, the work provided in [7], [8], [9] and [10] provided us strong background knowledge. In [7] the structure of the optimizer is presented along with discussion on the main issues handled by each optimizer module**.** Chaudhuri S. in [9] discusses how a query optimizer selects the best execution plan for a given query. Furthermore, fundamental requirements for search spaces, accurate cost estimation technique and best execution plan out of many candidates plans have also been discussed in detail.

III. ORGANIZATION OF THE PAPER

The rest of the paper is organized as follows: Section IV gives brief introduction of SQL, relational algebra and translation of SQL queries into relational algebra. Section IV also presents optimization strategies that we have focused on. Section V gives tool details and discusses experimental results. Finally, section VI concludes the work and gives suggestions for the future work.

IV. QUERY TRANSLATION AND OPTIMIZATION

A. *Structured Query Language*

Structured query language (SQL) is declarative query language developed for users' comfort that tells the database what user wants.

Structure of SQL Query is based on three clauses:

SELECT $\text{Column}_1, \text{Column}_2, \ldots, \text{Column}_n$
FROM $\text{Table}_1, \text{Table}_2, \ldots, \text{Table}_n$
WHERE Condition

Consider the query which asks to ***"return the names of students who earned grade 'B' "***. This query is easily understood by a person but DBMS requires it to be represented in a language also understood by the DBMS. The relational database management systems allow users to enter queries in SQL. Above query can be represented in SQL as follows where "**Students**" and "**Enrolled**" are the names of relations:

SELECT Name
FROM Students, Enrolled
WHERE Students.id= Enrolled.id
AND Enrolled.Grade='B'

Since SQL queries have been developed for the ease of users, DBMS translates SQL queries in relational algebra in order to access the data.

B. *Relational Algebra*

As opposed to SQL, relational algebra is procedural language that not only tells the DBMS ***what*** user wants but also tells ***how*** to compute the answer. Relational algebra is based on relations and operators that operate on relations; see Fig. 1. Most commonly used relational operators are:

- **Select (σ):** Returns tuples that satisfy a given predicate (condition or formula).
- **Project (π):** Returns attributes listed
- **Join (⋈):** Returns a filtered cross product of its arguments.

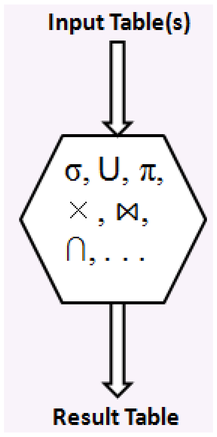


Fig. 1. Relational operators take one or two relations as input and produce a resultant relation.

C. *SQL to Relational Algebra Translation*

In order to translate a SQL query into relational algebra, query processor translates each **SELECT** clause to **Projection (π), FROM** clause to **table name(s)** or their **Cartesian**, and **WHERE** clause to **Selection (σ).**

The query mentioned earlier to ***"return the names of students who earned grade 'B' "*** can be written in relational algebra as follows:

$\boldsymbol{\pi}_{\text{Name}} (\boldsymbol{\sigma}_{\text{Students.id = Enrolled.id} \wedge \text{ Enrolled .Grade = 'B'}} (\text{Students} \times \text{Enrolled}))$

However, query optimizer produces other equivalent alternative execution plans implemented in it so that a better plan in terms of execution time and resources may be found.

D. *Generation of Query Execution Plans*

After the query processor translates given SQL statement to relational algebra it forwards that expression to query optimizer which generates various executions plans representing different orders or combinations of operators.

There are a number of algebraic laws implemented in query optimizer for generating equivalent (logical) query plans. However, following are the most commonly used techniques that we also consider and implement in our tool: Simple, Elimination of Cartesian product, and Push selection.

- *Simple Execution Plan*

Query processor generates an equivalent relational algebraic expression for the input query and forwards it to the query optimizer. The first algebraic expression generated by the query processor involves Cartesian product that we call simple execution plan.

Example 1: ***return names of students who earned grade 'B'***

In SQL it can be represented as:

SELECT Name
FROM Students, Enrolled
WHERE Students.id= Enrolled.id
AND Enrolled.Grade='B'

In relational algebra above SQL statement is represented as: $\pi_{\text{Name}}(\sigma_{\text{Students.id = Enrolled.id} \wedge \text{Enrolled .Grade = 'B'}}(\text{Students} \times \text{Enrolled}))$

Fig. 2 shows the algebraic tree of the above expression:

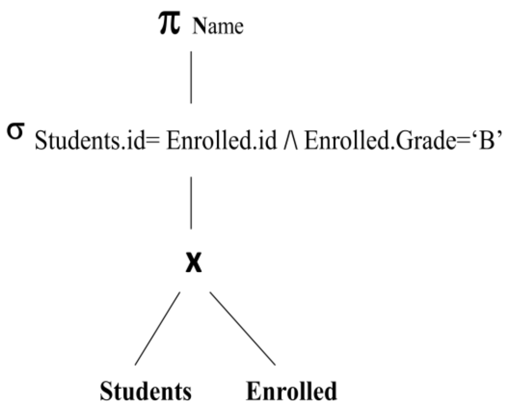


Fig. 2. Simple execution plan involving Cartesian product

- *Elimination of Cartesian Product*

Cartesian product operations can be combined with selection operations (and sometimes, with projection operations) which use data from both relations to form joins.

After replacing Cartesian product with Join, relational algebra for the query given in example 1 can be represented as:

$\pi_{\text{Name}}(\sigma_{\text{Enrolled .Grade = 'B'}}(\text{students} \bowtie_{\text{Students.id = Enrolled.id}} (\text{Enrolled})).$

Fig. 3 below gives the operator tree of the above algebraic expression:

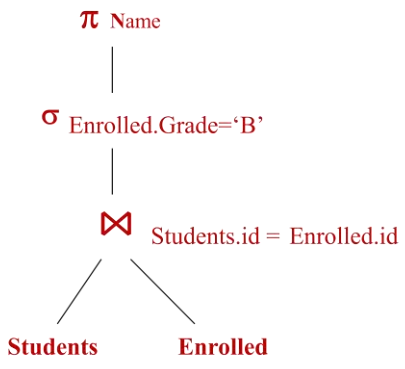


Fig. 3. Execution plan using Join

- *Push Selection*

Selections can be pushed down the expression tree as far as they can be pushed. By pushing selection operation down, we are actually decreasing the size of relations with which we need to work earlier.

Relational algebra for the query written in example 1 under push selection strategy can be represented as:

$\pi_{\text{Name}}(\text{students} \bowtie_{\text{Students.id = Enrolled.id}} (\sigma_{\text{Enrolled .Grade = 'B'}}(\text{Enrolled})))$

Fig. 4 below is the operator tree of the above mentioned relational algebraic expression:

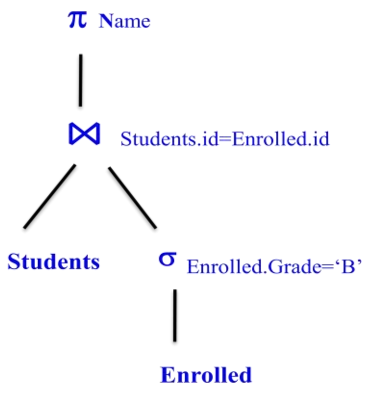
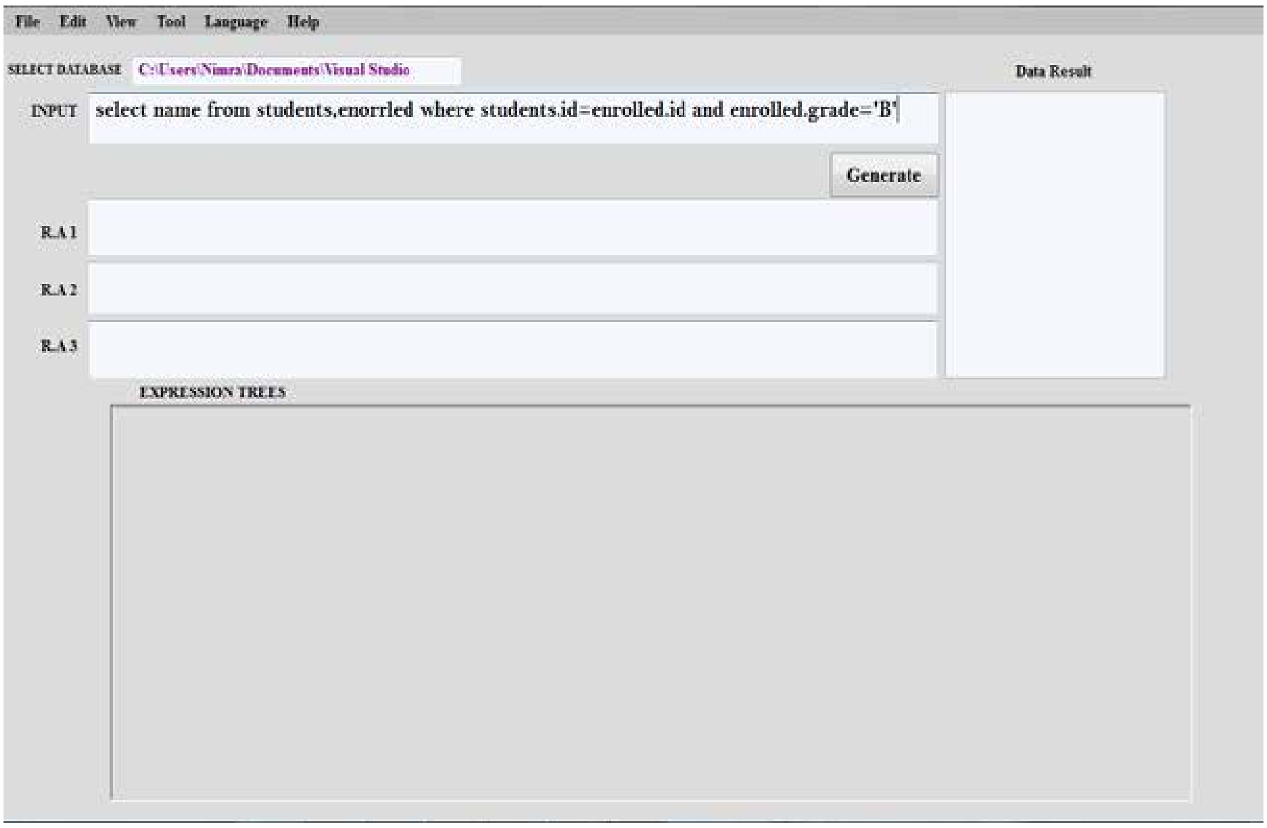


Fig. 4. Pushing selection down the tree

V. Tool Details and Results

A. *The Tool*

The developed tool gets user queries in SQL and provides equivalent relational algebraic expression. Furthermore, it generates equivalent execution plans under ***simple***, ***join*** and ***push-selection*** strategies and generates also their execution time. The user interface of the tool has been developed in C# and the database connectivity has been provided using SQL Server. Fig. 5 below gives the interface of the tool:



SELECT DATABASE C:\Users\Nimra\Documents\Visual Studio

INPUT select name from students,enorrled where students.id=enrolled.id and enrolled.grade='B'

R.A 1

R.A 2

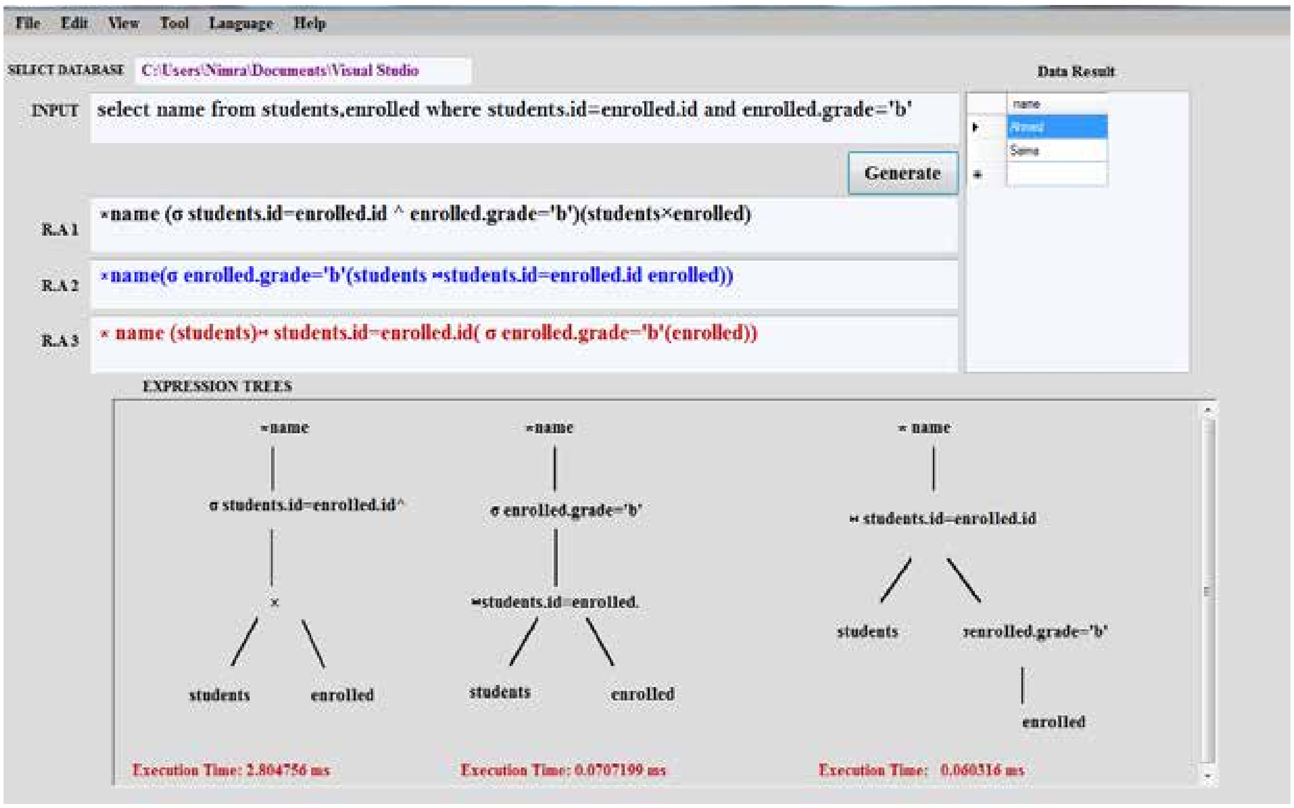
R.A 3

Fig. 5. User Interface of the tool with database connectivity and query input.

As shown in Fig. 5 the tool allows user to connect with a database using "Select Database" option on the top left.

After the database name has been provided user can write SQL query in input box, below the "Selection Database" option, as shown in Fig. 5.

Once the query has been provided, user presses



SELECT DATABASE C:\Users\Nimra\Documents\Visual Studio

INPUT select name from students,enrolled where students.id=enrolled.id and enrolled.grade='b'

	name
▸	Ahmed
	Sama
*	

R.A 1 ×name (σ students.id=enrolled.id ^ enrolled.grade='b')(students×enrolled)

R.A 2 ×name(σ enrolled.grade='b'(students ⋈students.id=enrolled.id enrolled))

R.A 3 × name (students)⋈ students.id=enrolled.id(σ enrolled.grade='b'(enrolled))

Execution Time: 2.804756 ms

Execution Time: 0.0707199 ms

Execution Time: 0.060316 ms

Fig. 6. Generation of equivalent algebraic expressions and operator trees implementing simple, elimination of Cartesian product with join and push selection strategies and their execution time.

generated along with their operator trees and execution times, see Fig. 6.

As shown in Fig. 6 the tool also generates the result of the computation in "data result" portion at top right. The execution times are obtained using SQL server data computation library.

B. Results and Discussion

As shown in Fig. 6. Simple execution plan seems to be more expensive as it takes around 2.8 milliseconds to get the result. Push selection on the other hand seems to be taking less time compared to other two execution plans. In order to justify our results, let us consider two relations Student and Enrolled given in TABLE I and TABLE 2 respectively:

TABLE I. STUDENT

id	**Name**
11	Ali
22	Ahmed
33	Saima

TABLE II. ENROLLED

id	**C.no**	**Grade**
11	CS20	A
22	CS21	B
33	CS22	B

Now let us consider the query:

```
SELECT   Name
FROM     Students, Enrolled
WHERE    Students.id= Enrolled.id
AND      Enrolled.Grade='B'
```

Under simple execution plan, algebraic expression is:
π_{Name} ($\sigma_{\text{Students.id = Enrolled.id} \wedge \text{Enrolled .Grade = 'B'}}$ (Students × Enrolled))

Operator trees are executed in bottom to top approach Therefore, first of all the Cartesian between two tables (**Students**, **Enrolled**) is performed, then selection is performed and finally projection is performed. The result of **(Students × Enrolled)** is provided in TABLE III.

TABLE III. RESULT OF CARTESIAN PRODUCT BETWEEN STUDENT AND ENROLLED

id	**Name**	**id**	**C.no**	**Grade**
11	Ali	11	CS20	A
11	Ali	22	CS21	B
11	Ali	33	CS22	B
22	Ahmed	11	CS20	A
22	Ahmed	22	CS21	B
22	Ahmed	33	CS22	B
33	Saima	11	CS20	A
33	Saima	22	CS21	B
33	Saima	33	CS22	B

After performing Cartesian the next operation performed is selection on resultant table. The expression: $\sigma_{\text{Students.id=Enrolled.id} \wedge \text{Enrolled. Grade ='B'}}$ computes to TABLE IV below:

TABLE IV. TABLE OBTAINED AFTER SELECTION OPERATION PERFORMED ON TABLE III

id	Name	id	C.no	Grade
22	Ahmed	22	CS21	B
33	Saima	33	CS22	B

Finally, the last operation performed is projection operation $\pi_{\textbf{Name}}$ on TABLE IV. The result of the projection operation is given in TABLE V.

TABLE V. RESULT OBTAINED AFTER πNAME

Name
Ahmed
Saima

From this computation it can easily be inferred that this query cannot be the best way to reach the final answer; because of cross product. This means that we create a table whose number of rows is **|Studentst|** * **|Enrolled|**.

One of the other possible ways of improving above computation is to replace Cartesion product with Join operator followed by the selection. Experession with Cartesian replaced with join is shown below:

$\pi_{\text{Name}} (\sigma_{\text{Enrolled .Grade = 'B'}} (\text{students} \bowtie_{\text{Students.id = Enrolled.id}} (\text{Enrolled}))$

Under this expression the first operation performed is the Join operation on Students and Enrolled tables: $\text{Students} \bowtie_{\text{students.id = Enrolled.id}} (\text{Enrolled})$

The result of performing Join operation on Students and Enrolled tables for common IDs is shown in TABLE VI.

TABLE VI. RESULT OF JOIN OPERATION ON STUDENTS AND ENROLLED TABLES

id	Name	id	C.no	Grade
11	Ali	11	CS20	A
22	Ahmed	22	CS21	B
33	Saima	33	CS22	B

After the Join, the next operation performed is selection $\sigma_{\text{Enrolled. Grade ='B'}}$ whose result is given in TABLE VII.

TABLE VII. RESULT OF SELECTION OPERATION PERFORMED ON TABLE VI

id	C.no	Grade	id	Name
22	CS21	B	22	Ahmed
33	CS22	B	33	Saima

Finally, the last operation performed is the projection π_{Name} whose is result is provided in TABLE VIII below:

TABLE VIII. TABLE OBTAINED AFTER PROJECTION OPERATION ON TABLE VII

Name
Ahmed
Saima

Query under Join operation took less execution time as compared to first plan because the number of rows has decreased.

The third query plan we have considered is push selection. The algebraic expression under this strategy is: $\pi_{\text{Name}}(\text{Students} \bowtie_{\text{Students.id=Enrolled.id}}(\sigma_{\text{Enrolled.Grade='B'}}(\text{Enrolled})))$.

According to this expression, selection operation $\sigma_{\text{Enrolled.Grade='B'}}(\text{Enrolled})$ is performed before Join and projection operations. The result of the selection operation is given in TABLE IX below:

TABLE IX. RESULT OF SELECTION OPERATION ON ENROLLED TABLE

id	C.no	Grade
22	CS21	B
33	CS22	B

The next operation performed is the Join operation of the STUDENTS TABLE (TABLE I) and the TABLE IX. The resultant table is shown below:

TABLE X. RESULT OF THE JOINT OPERATION

id	C.no	Grade	id	Name
22	CS21	B	22	Ahmed
33	CS22	B	33	Saima

Finally the last operation performed is projection π_{Name} whose is result is given in TABLE XI below:

TABLE XI. RESULT OF THE PROJECTION OPERATION

Name
Ahmed
Saima

From this computation, we learn that earlier we process selections, less tuples we need to manipulate higher up in the tree and thus less execution time is required.

VI. CONCLUSION

In this paper, we presented a tool that translates user queries (entered in SQL) into mathematical representation (relational algebra) necessary for data access. It has been proved that the developed tool is better at: producing syntax and semantic errors, generating query execution plans: (i) Simple, (ii) elimination of Cartesian product with join, and (iii) push selection, allowing connectivity with databases of interest and produces query results.

Query optimizer of the tool produces execution times of all three execution plans considered. Based on the obtained results, it is concluded that push-selection is better than the join and join is better than Cartesian product in terms of execution time.

Tool may be used (in both academic and research institutes) for better understanding of how queries are actually processed and optimized and also for the development of modern DBMS packages which will be more efficient in terms of execution time.

FUTURE WORK

The measurement of the consumption of resources, Analysis of the tradeoff between execution time and consumption of resources, and implementation of few other optimization techniques are planned for the future.

ACKNOWLEDGMENT

This paper is based on our Masters' thesis defended in February 2015 at Quaid-e-Awam University of Engineering, Science & Technology, Nawabshah, Pakistan.

REFERENCES

[1] Elmasri R., and NavathS. B. "Fundamentals of database systems", isbn: 978-81-317-1625-0, Addison-Wesley Longman Publishing Co., Inc. 2010.

[2] Bernstein P. A., and Newcomer E. "Principles of Transaction Processing". isbn: 9780080948416, 1997.

[3] D. Kossmann. "The State of the Art in Distributed Query Processing". ACM Computing Surveys, vol. 32, pp 422–469, isbn 0360-0300, Dec. 2000.

[4] Stefano Ceri, Georg Gottlob, "Translating SQL Into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries", Software Engineering, IEEE Transactions, vol. SE-11, issue 4, pp. 324 – 345, Apr. 1985

[5] XU Silao,HONG Mei,"Translating SQL Into Relational Algebra Tree-Using Object-Oriented Thinking to Obtain Expression Of Relational Algebra", IJEM, vol.2, no.3, pp.53-62, 2012.

[6] Bendre M. "Relational algebra translator", 2013. http://www.slinfo.una.ac.cr/rat/rat.html.

[7] Y. Ioannidis, "Query Optimization", *Journal ACM Computing Surveys (CSUR)*, vol. 28, issn 0360-0300, Mar. 1996.

[8] Santhi Lasya "A Study of Library Databases by Translating Those SQL Queries Into Relational Algebra and Generating Query Trees" *International Journal of Computer Science Issues(IJCSI)*, vol. 8, Issue 5, No 1, Sep. 2011.

[9] Chaudhuri S. Mendelzon A. and Paredaens J. "An overview of query optimization in relational systems". *In proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 34–43, isbn: 0-89791-996-3, Jun 1998.

[10] Majid Khan and M. N. A. Khan. "Exploring Query Optimization Techniques in Relational Databases". *In proceedings of the International Journal of Database Theory and Application.* vol. 6, No. 3, Jun. 2013.