Research Journal of Information Technology

ISSN 1815-7432

Academic Journals Inc.

www.academicjournals.com

Research Journal of Information Technology 2 (1): 11-20, 2010
ISSN 1815-7432

Novel Strategies to Speed-Up Query Response

Md. T. Hoque and V.M. Avery
Discovery Biology, Eskitis Institute for Cell and Molecular Therapies,
Brisbane Innovation Park Campus, Nathan, Griffith University,
QLD-4111, Australia

Abstract: Intuitive but completely novel strategies have been proposed which provides noteworthy improvement in reducing the query response time, assuming a database is designed in third or higher normal form to avoid obvious design pitfalls. In query construction, the cartesian product has been replaced by a novel substitution approach, which reduces both the query execution time and space many order magnitudes compared to the traditional approaches, along with other benefits. We had implemented many database-applications with the proposed paradigm to cope with the practical bottlenecks of the database-systems. Hence the proposed paradigm had been rigorously and positively tested for its practical usefulness. The proposed substitution based paradigm, compared to the traditional query execution by Cartesian approach, performs faster and effective having both time and space complexities reduced significantly.

Key words: Speed up query, normal form, substantiation approach, cartesian product

INTRODUCTION

The use of database in a client-server environment is a very common practice now-a-days for most of the organizations. It is not very easy to fit the client-server based application satisfying the entire requirements perfectly especially while the database becomes larger and complicated. Also, a lot of factors are involved with the acceptance of the application software to the users. One of the vital factors is the level of computer-literacy of the users which affect their acceptance of the application software. Well-designed database application software must also consider resource constraints. Further, it is also required to consider the design to be fault-tolerant against the known and unknown bugs and limitations of the database server, system and other components, with respect to hardware and software as well as the network environments. Even any minor overlooked-anomaly in design would scale up with the growing needs and size of the database causing any future alterations costly and labor intensive. Therefore, database software developers need to consider countering as many limitations as possible earlier and should consider the database based application optimistically as well. This study is intended to site few special limitations countering requirements that arose from the practical cases we experienced in study.

Corresponding Author: Dr. Md Tamjidul Hoque, Discovery Biology,
Eskitis Institute for Cell and Molecular Therapies,
Brisbane Innovation Park campus, Nathan, Griffith University,
QLD-4111, Australia Tel: +61-7-37356040 Fax: +61-7-37356001

Query optimization and related issues are one of the most visited and effort intensive area (Babcock and Chaudhuri, 2005; Chang and Lee, 1997; Chaudhuri *et al*., 1998; Chaudhuri, and Shim, 1999; Chaudhuri and Narasayya, 2001; Das and Batory, 1995; Galindo-Legaria and Rosenthal, 1997; Hellerstein, 1998; Menascé and Gomaa, 2000; Stocker *et al*., 2001; Yi-Leh *et al*., 2002; Ya and Egyhazy, 1997) in Database Management System (DBMS) by the research community. From the practical perspective, it has been seen that the updated versions of database-systems or DBMS including the improved query optimizer does not perform significantly as expected (Das and Batory, 1995). With the increasing size of the database in every aspect of meeting the ever-sophisticated requirements for a fast growing company, our study found that in practical case speeding up crisis for query-response becomes an adamant requirement. To cope, a number of usual steps are taken, such as:

- Adding and updating index or indexing the database
- Updating the version of the database
- Redesign to fit the updated connecting component (such as converting from RDO to ADO)
- Upgrading network components such as powerful and sophisticated switch/routers
- Hardware upgrading such as usage of multiprocessors, more powerful processors, using SCSI hard disk drive instead of IDE and so on

The aforementioned costly and cumbersome steps are usually performed; however, those steps do not reflect much convincing improvement in reducing the increasing query response time due to the increasing load and size of the database. We had been confronting with the above-mentioned challenge to be resolved satisfactorily as we were dealing with running many (around 70) real-world different database-applications within the very large company having many transactions, users and clients. Those applications are mostly larger in size ranging from around 10 to 900 MB (at the initial loading) and on average of having size of 600 MB. This critical situation encouraged us or more frankly pushed us to solve these problems.

In this study, we showed a completely new approach, which to the best of our knowledge has never been reported earlier. The standard query writing strategies using *Cartesian* product has been replaced by our proposed substitution approach. This provides a number of benefits such as, most importantly:

- A significant improvement in the query response time
- Require very less query execution space
- Reduces network traffic
- Urgency of the costly upgradation of the hardware including network and software is minimized
- Further, in the client-server architecture the loads are distributed and balanced
- It can be implemented at the front end by the developer directly
- The approach also removes vulnerable RDO connectivity component failure, which is a common victim of query time out for the query performed on a moderately sized database such as 100 MB or above (using MS SLQ 6.5) and so on

The requirement of the database design in higher normal form to implement our strategies is increasingly supportive and minimum 3NF is expected, which implies our approach can be implemented to reasonably pitfall free all databases and hence, the impact can be noteworthy with the superior improvement in the query response time.

DEFINITATIONS AND THE CONTEXT

To understand the paradigm and the context used in this study, followings definitions and points are discussed.

Normal forms-normalization has great importance (Silberschatz *et al*., 2002; Howe, 2001) to avoid any pitfall while designing database. Normalization is a design technique (Kline *et al*., 1999) that enables database designers and programmer to develop table structures that minimize programming problems. Levels of adherence called normal form characterize the process of normalization. If a table design conforms only to lowest level of normalization, it is said to be in First Normal Form, which is abbreviated as 1 NF. A table having first normal form has atomic (Rensin and Fedorchek, 1997) values and a primary key and the order of its rows and columns must not matter. If a design table conforms to the next higher level, it is in Second Normal Form (2 NF). The second normal form requires that all non-key columns depend on the whole primary key. Therefore, if the primary key of a table is in a single column, it is automatically in the second normal form. The Third Normal Form (3 NF) requires that there need to have no transitive dependencies among non-key column. This means that we cannot have one non-key column dependent on another non-key column for its definition.

The collective experience (Ioannidis, 1996; Hoque, 2003; Silberschatz *et al*., 2002) of database designing over the years shows that using the 3 NF is sufficient to avoid the vast majority of pitfalls in data management and the 3 NF is heavily used for a stable design, where the higher normal guarantee further problem free design but rarely required to use which is basically breaking a table further to segregate any possible functional dependencies to eliminate redundancy, organize data efficiently and avoids any possible anomalies.

To understand the normalization effect, consider a simple example assuming that we are keeping an employee database for the group of companies and each employee baring a Personal File Number (PF_No.) is associated with any one of the units of the group from where the employees draw their salaries as indicated in Table 1.

Now consider the (Salary) Unit-name referred in Table 1 and say SNIL has been changed to HSNIL as decided by the company. Then it will be required to change in all possible places, wherever the SNIL exist, converting them into HSNIL in Table 1. The updating would be expensive and vulnerable to anomalies for a complicated database with many tables having the unit-name in this fashion. If normalization is applied, the above Table 1 can be broken down into the following two tables namely Table 2 and 3. Unit_Id field (i.e., Foreign Key) of Table III refers to the Unit_Id field of Table 2. Now, it is easy to change SNIL to HSNIL in only one table, namely Table 2, which will solve the purpose.

Table 1: Employee list with their associated salary unit

PF_No.	Employee-name	Salary unit-name	Joining date
2034	Taysir	EWPD	15/12/2002
4512	Rezual	BPML	20/06/2009
5234	Shabbir	SNIL	24/09/2004
6456	Abeed	BPML	25/10/2006
7235	Kabir	SNIL	02/02/2008

Table 2: Unit name of the group of companies

Unit_Id	Salary unit name
1	EWPD
2	BPML
3	SNIL

Table 3: Employee details with associated salary unit

PF No.	Employee name	Unit Id	Joining date
2034	Taysir	1	15/12/2002
4512	Rezual	2	20/06/2009
5234	Shabbir	3	24/09/2004
6456	Abeed	2	25/10/2006
7235	Kabir	3	02/02/2008

Query Optimizer

The query optimizer is the component of a DBMS (Ioannidis, 1996; Silberschatz *et al*., 2002) that through its query plans (mostly as tree) attempts to determine the most efficient way to execute query. For the cost base approach, costs are assigned to possible path to execute the query and the least cost is preferred. In terms of I/O operation required, CPU time (Chaudhuri *et al*., 1998; Chaudhuri and Narasayya, 2001) and resource allocation, search space so on are considered for costing purpose. The simple idea for query optimization is mostly based on join order in practice using a dynamic programming (Stocker *et al*., 2001). The performance of a query plan is determined largely by the order (Yi-Leh *et al*., 2002; Galindo-Legaria and Rosenthal, 1997) in which the tables are joined. For example, when joining 3 tables X, Y, Z of size 20, 30000 and 20000 rows, respectively, a query plan that joins Y and Z first can take several orders of magnitude of more time and space to execute than one that joins X and Z first.

Type of database tables and uses-For the 3 NF or higher normal form such as Boyce-Codd Normal Form (BCNF) or higher (i.e., 4NF, 5NF, etc.), the tables can be divided into two major category-one is the data table containing the concerning data such as Table 3. Let them call Informative Data Table or IDT in short. Usually the number of rows and growth of the rows for this kind of table is very high. Relatively they have far more number of columns than the other type. Other type table is to help the IDT to define its referencing column(s), let this group be called Meta Data Table or MDT in short. Table 2 is an example of MDT. The growth of the MDT in terms of row is usually very low and numbers of rows and the columns both are also very low. But, among total number of tables the MDTs are very high in number compared to the number of IDTs in a database.

On the other hand, the database users can be divided broadly into two types in categories namely, the Online Transaction Processing (OLTP) system users and the Decision Support System (DSS) system users. An OLTP system, as the name implies, processes transactions interactively to update a database with some insertion and deletions (Austin *et al*., 1999). In OLTP system, a lot of random reads and writes are expected. On the other hand, DSS system supports analysis of the database. The DSS usually has a lot of complex queries, both in terms of joins and in terms of aggregating data. In DSS system, almost no transaction type updating is expected, but very bulk data loading is highly likely. Under resource constraint, same server machine is used for OLTP and DSS system instead of using minimum of two separate machines for two different system users' group.

The DDS type users are very influential in the sense that their requirements largely affect the database application design and development. The DSS users are very important for taking major business decisions. Most of the cases DSS users like to bring a snapshot of the information by choosing few major conditions and later like to have sub-queries on any subset of the information, i.e., zooming into the snapshot in various angles. For example, a marketing manager, first like to search, how many clients are there whose number of monthly installment dues are more that fifteen. Later upon having the query results, he might like to find, how many of them are under new project schedule or under delayed installment schedule. And then he issues command selectively to send emails to different group having different categories of automated reminder. Or, he further searches from the snapshot,

whether there is any potential client left in the defaulter-list to be removed for not sending the client a warning email or, letter for the sake of maintaining future business relations and so on.

IMPROVED DESIGN PARADIGM SUPPORTING FASTER QUERY RESPONSE USING CASE STUDY

As a continuation of the context of the above example, when the manager zooms into the snapshot of the bulk initial queried results, if further fetching is done from the database server, the delay involved due to resource constraints (assume network is having heavy traffic or the infrastructure is not costly) along with re-fetching can make the manager loses focus for taking decisions in reality. While we were investigating our speeding up approaches, we found that it is satisfactory for the DSS users to fetch the sub-query or snap from local/client machine. Thus, to have resolution in supporting the issue, novel design paradigm has been initiated focusing the client side (front-end) application-development (Menascé and Gomaa, 2000). The bulk or larger snapshot is first collected from server along with other possible side-result requirements and store into local machine using local database tables temporarily. To do so, often denormalization of a table, i.e., converting higher normal form back to a lower normal form, such as 3NF to 2NF and needs be stored in the local database table temporarily. This can have other benefit such as improved presentation, which can easily be maintained with the local database concept. With these novel applications, the sub-queries are speeded-up and the DSS users are satisfied with the change. They even do not mind to wait a bit more for properly loading more meaningful initial snapshot. Later they can easily concentrate on the sub-queried results. Since, the re-fetching from the server is reduced, it also reduces redundant fetching and reduces the network traffic. Thus the overall query processing time minimized by having non-redundant fetching. Furthermore, the OLTP users would face less frequent delays in posting or query-searching due to the lesser reflection activities of the DSS type users in the server as well as in the network.

To reduce the network traffic (Menascé and Gomaa, 2000), we have further used stored procedure, especially when one query-result is returned to generate further query without the requirements of users' involvement or feedback in the middle. A stored procedure is a precompiled set of statements that can perform iterative conditional and set-based logic on the data in the database server. Stored procedure may have parameters passed to them only (this reduces network traffic since instead a large query statement, name of the Stored Procedure is send only with or without few parameters) and it may also return result sets. Semi-join reducer (Stocker *et al*., 2001) can be used as well.

Since, query is the heart of the activities of the database-system or DBMS, so in the designing level, query-designing and optimization should get special attention (Menascé and Gomaa, 2000). A query execution delay or execution failure may not be differentiable or visible for database having small-sized data. But when the database is possessing and processing enormous data, it becomes nightmare for the developer with the traditional design paradigms, facing many critical problems and complains and required to solve them within a very short time. The problems are like: query time out ..., tempdb size needs to be increased ..., (Microsoft Corporation, 1999) and so on; these problem-messages from the database-systems keep the developer annoyed. Often it requires developer to spend nights to convert the whole program-code to fit ADO-OLEDB combination instead of RDO-ODBC combination and further updating the version of ADO, or setting infinite

```
select *
from T1, T2, ..., Tn
where ... ... ... ...
```

Fig. 1: A query outline

timeout to the connection properties, indexing the database tables and so on. Designing a database structure theoretically and implementing a database structure practically could be two different tasks (Adams and Beckett, 2009) due to bugs and scaling-limitations of the database-systems. Thus, implementing a structure without properly revising a designing to cope the limits would lead to a non-robust development. As a pitfall, for the non-robust system and design, it becomes urgent to modify when previously mentioned problems become apparent but as the growing database is in use, it gets increasingly difficult and costly to debug and to solve the problems. We are motivated to provide a better solution to all these confrontations as discussed in details next, starting from query optimization.

Most of the case a query with large number of tables do cartesian product (Zinke, 2009) or are crossed where the IDT(s) are having relatively large number of rows-leading to query failure or taking a great deal of time to return the results. To deep into the matter, let us investigate the following query outlined as:

From the second line of query, it is found that table T_i, where, i = 1 through n, are crossed. In the design having 3NF or higher normal form, the number of IDTs are usually relatively much lesser than MDT in the queries, for example, if n = 6, it is very likely that there are one IDT and 5 MDTs. Assume in a query, outlined as shown in Fig. 1 that T_1 is the IDT and all the other tables are MDTs and n = 4. Assume, T_1 has total of 1,000,000 (One million) rows and i = 2 through 4 for T_i each having 250 rows on the average. Also, assume the where clause of the query would match 100 rows from T_1.

We like to investigate what would the query execution like Fig. 1 would perform to formulated the query result: At the server end, for an estimation the cartesian product can be worked out as $(T_1 \times T_2 \times T_3 \times T_4)$ and the number of rows of the product or the crossing would result as, $(1000000 \times 250 \times 250 \times 250) = 15625,000,000,000$ rows and those required to be temporarily stored at the server end and then fetch 100 out of 15625,000,000,000 rows for the client end, in the worst case. Certainly, a larger space required at the server side to ensure the execution and that is vulnerable to be failure attempt, specially, when free space become too short at the server end, also the time (Keller and Clodewey, 2009) need to be considered which involved in producing the result from the cartesian product. Not only this, it is very frightening to think of the situation where more than one user is submitting the similar query at the same time and the number of such users is high.

If we investigate deeper and look closer, we see that we require only 100 of 1,000,000 rows of T_1. In this case, we can fetch and load the smaller MDTs (T_i, where, i = 2 to 4) at the client end or front end and then fetch directly using modified and simplified query, the 100 rows of T_1 from the database server. And we can substitute the referencing columns of the 100 rows of the T_1 by corresponding preloaded MDTs while placing them into the temporary local database at the client side. The benefits are, almost no additional free space is required at the server side, the power of the client machine is being utilized and i.e. the client-server load is balanced and the execution-time is reduced heavily. For example, if we want to list those employees associated with the EWPD unit, using the Table 2 and 3 the straightforward query would be as Fig. 2.

```
select PF_No, EmployeeName, SalaryUnitName, JoiningDate
from Table3, Table2
where Table3.Unit_Id = Table2.Unit_Id
and SalaryUnitName = 'EWPD'
```

Fig. 2: Straightforward query for searching the Employee list of EWPD unit

```
select *
from Table3
where Unit_id = 1
```

Fig. 3: Non-straightforward query for searching the Employee list of EWPD unit

At first for the crossing, temporary table of 15 or, (5×3) rows are created. Among the rows, the where clause would qualify 1 raw and return. But with modified query strategy, namely the substitution approach, Table 2 would be copied and pre-loaded to the client end. Later the following query of Fig. 3 would be executed.

No crossing or cartesian product is done; the qualified one row is fetched and returned direct and further at the front end or the client side the value of Unit_id = 1 is substituted by EWPD from the pre-loaded copy of Table 2. If it is noted carefully, another benefit is visible. In Fig. 3 the comparison under 'where' clause (i.e., Unit_id = 1) is numeric comparison, whereas in Fig. 2, it is a string based comparison. Furthermore, a numeric comparison is faster than string comparison for obvious reason.

RESULTS

We ran the experiments on around 70 real-life databases of various sizes from 10 to around 850 MB of size and having around 15 to 100 tables, developed for the group of companies. The experiment was carried out using two versions of SQL server namely, Microsoft SQL-6.5 (SQL6.5) and Microsoft SQL-2000 (SQL-2K). The RDO ODBC combination failed very often for the large sized database – therefore, for the comparison purpose we used ADO OLEDB combination only to compare the results, but it is also worth mentioning that the RDO ODBC never failed while applied with our proposed substitution approach. For all of the cases, all possible indexing are created and then work together with all default query optimizer of the database. The proposed Substitution Approach (SA) is used for both SQL versions are referred as SQL-6.5_SA and SQL-2K_SA.

We ran same query for the same database in four different conditions namely, SQL6.5, SQL-2K, SQL-6.5_SA and SQL-2K_SA. The involved IDT Table's size was proportional to the size of the database and the same run was repeated 100 times and the average response time (in second) was recorded. Though there were differences between SQL-6.5 and SQL-2K runs, but they both turned into exponential growth for the response time with respect to the growth of the database. On the other hand, response time of SQL-6.5_SA and SQL-2K_SA using proposed substitution approach were radically reduced compared to the traditional approach. Both SQL-6.5_SA and SQL-2K_SA showed linearity though there was a small difference between the responses of the two. Performance comparison between standard versus proposed substitution approach is shown in Fig. 4.

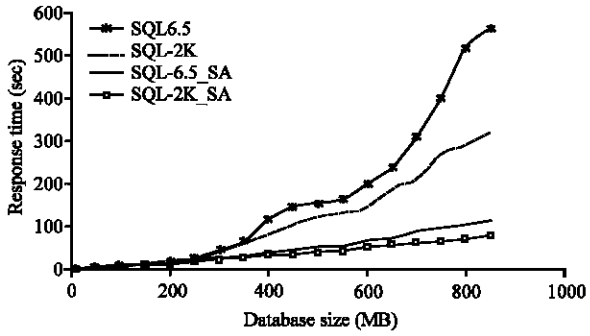

Fig. 4: Performance comparison between standard versus proposed substitution approach

DISCUSSION

To make the query execution faster many research-works have been exhaustively performed to minimize the query-execution path cost within the DBMS. However, a totally different and potential way of effective and efficient query-execution remains unexplored out of the traditions. We have introduced the novel approach in this article and most importantly our approach remains complementary to the traditional approach. Our proposed substitution approach based paradigm for obvious reason performs significantly faster and efficient in the query execution and to answer why, assume that there are, m tables having n rows each. The order of both time and space complexities of standard or traditional query execution by Cartesian product in the worst case is $O(n^m)$ whereas for the proposed substitution approach, it would take linearity pattern in the time and space complexities which is O(mn). Moreover, the exponential load is usually limited within the server keeping the client idle for the traditional approach whereas in the proposed approach there is a scope to establish the balance between the client and server regarding both time and space. The outcome reflected in better management for the DSS users by allowing local database within the proposed paradigm and the OLTP users where not affected with blocked or delayed transactions.

However, the pre-condition of using the proposed approach is that the database needs to be in higher normal form where minimum 3NF is expected. Clearly the requirement is complementary and supportive-since, to avoid any possible pitfall, higher normal forms are followed and the substitution approach is increasing applicable with higher normal form, therefore, our approach can be implemented design-pitfall-free (or, having 3NF or higher normal form) all the databases. And for obvious reason a design-pitfall-free database application is expected.

The proposed novel approach can be considered as the immediate upper layer of the query optimizer, where it can be easily placed. The query optimizer has been developed, as DBMS component with the philosophy of no user intervention be required-but the proposed substitution approach that we have developed, is at the design level and accessible and implementable by the designer. The database designer now can easily incorporate these ideas into new designs, however, for existing applications, alteration cost might be a factor, which still can be well compensated by not spending on traditional and less effective counter steps as mentioned earlier. The strength of our claim and success is that we have implemented this approach towards many database applications and successfully removed the previously mentioned critical problems that we were practically confronted with and we rather improved the performance heavily. Therefore, we believe the achievements are very

meaningful and worthy to the database-application developer and designer and could be more meaningful if the concept can be applied within the Database Management Systems (DBMS) in the near future.

REFERENCES

Adams, D. and D. Beckett, 2009. Normalization is a Nice Theory: 4D Summit. Foresight Technology Inc., USA.

Austin, D., W. Baird, M. Burke, N. Chase and J. Duer *et al*., 1999. Special Edition Using Oracle8/8i. QUE., USA.

Babcock, B. and S. Chaudhuri, 2005. Towards a robust query optimizer: A principled and practical approach. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 14-16, ACM. New York, USA., pp: 119-130.

Chang, J. and S. Lee, 1997. An optimization of disjunctive queries: Union-pushdown. Proceedings of 21st Annual International Conference on Computer Software and Applications, Aug. 11-15, Washington, DC. USA., pp: 356-361.

Chaudhuri, S., 1998. An overview of query optimization in relational systems. Proceedings of ACM SIGACT-SIGMOD-SIGACT Symposium on Principles of Database Systems, June 1-4, ACM, New York, pp: 34-43.

Chaudhuri, S. and K. Shim, 1999. Optimization of queries with user-defined predicates. ACM Trans. Database Syst., 24: 177-228.

Chaudhuri, S. and V. Narasayya, 2001. Automated statistics management for query optimizers. IEEE Trans. Knowledge Data Eng., 13: 7-20.

Das, D. and D. Batory, 1995. Prairie: A rule specification framework for query optimizers. Proceedings of the 11th International Conference on Data Engineering, March 6-10, IEEE. Xplore, pp: 201-210.

Galindo-Legaria, C. and A. Rosenthal, 1997. Outerjoin simplification and recording for query optimization. ACM Trans. Database Syst., 22: 43-74.

Hellerstein, J.M., 1998. Optimization techniques for queries with expensive methods. ACM Trans. Database Syst., 23: 113-157.

Hoque, M.T., 2003. A case study: Few non-straight forward design approach for the client-server based database environment. Technical Report TR-2003/2, IT, Bashundhara Group.

Howe, D.R., 2001. Data Analysis for Data Base Design. 3rd Edn., Butterworth-Heinemann, USA.

Ioannidis, Y.E., 1996. Query optimization. ACM Comput. Surveys, 28: 121-123.

Keller, W. and J. Clodewey, 2009. Relational Database Access Layers: A Pattern Language. Software Design and Management GmbH and Co., USA.

Kline, K., L. Gould and A. Zanevsky, 1999. Transact-SQL Programming. 1st Edn., Oreilly and Associates Inc., Philadelphia.

Menascé, D.A. and H. Gomaa, 2000. A method for design and performance modeling of client/server systems. IEEE Trans. Software Eng., 26: 1066-1085.

Microsoft Corporation, 1999. Administrator's Companion, Microsoft SQL Server Version 6.0, for the Microsoft Windows NT Operating System. Microsoft Co., Singapore.

Rensin, D.K. and A.M. Fedorchek, 1997. SQL Server 6.5 Secrets. 2nd Edn., COMDEX Computer Publishing, New Delhi.

Silberschatz, A., H. Korth and S. Sudarshan, 2002. Database System Concept. 4th Edn., McGraw-Hill, Inc., USA.

Stocker, K., D. Kossmann, R. Braumandl and A. Kemper, 2001. Integrating Semi-Join-Reducers into State of the Art Query Processors. Proceedings of the 17th International Conference on Data Engineering, April 02-06, IEEE Computer Society Washington, DC, USA., pp: 575-584.

Ya, Z. and C. Egyhazy, 1997. Developing a query optimizer for a federated database system. Proceedings of IASTED International Conference on Intelligent Information Systems (IIS), Dec. 08-10, Grand Bahama Island, BAHAMAS., pp: 420-427.

Yi-Leh, W., D. Agrawal and A. El-Abbadi, 2002. Query estimation by adaptive sampling. Proceedings of 18th International Conference on Data Engineering, Feb. 26-March 01, San Jose, CA, USA., pp: 639-648.

Zinke, F., 2009. Relational database: A practical foundation. http://ddi.cs.uni-potsdam.de/Lehre/TuringLectures/Papers/Zinke.pdf.