On Using an Automatic, Autonomous and Non-Intrusive Approach for Rewriting SQL Queries

Arlino H. M. de Araújo[1,2], José Maria Monteiro[2], José Antônio F. de Macêdo[2], Júlio A. Tavares[3], Angelo Brayner[3], Sérgio Lifschitz[4]

[1] Federal University of Piauí, Brazil
[2] Federal University of Ceará, Brazil
{arlino,monteiro,jose.macedo}@lia.ufc.br
[3] University of Fortaleza, Brazil
brayner@unifor.br
[4] Pontifical Catholic University of Rio de Janeiro
sergio@inf.puc-rio.br

Abstract. Database applications have become very complex, dealing with a huge volume of data and database objects. Concurrently, low query response time and high transaction throughput have emerged as mandatory requirements to be ensured by database management systems (DBMSs). Among other possible interventions regarding database performance, SQL query rewriting has been shown quite efficient. The idea is to rewrite a new SQL statement equivalent to the statement initially formulated, where the new SQL statement provides performance gains w.r.t. query response time. In this paper, we propose an automatic and non-intrusive approach for rewriting SQL queries. The proposed approach has been implemented and its behavior was evaluated in three different DBMSs using TPC-H benchmark. The results show that the proposed approach is quite efficient and can be effectively considered by database professionals.

Categories and Subject Descriptors: H.2 [**Database Management**]: Miscellaneous

Keywords: Query processing, Database tuning, Query rewriting

1. INTRODUCTION

Database queries are expressed by means of high-level declarative languages, such as SQL (Structured Query Language), for instance. Such queries are submitted to the query engine, which is responsible for processing queries in database mangement systems (DBMSs). Thus, query engines should implement four main activities: query parsing, logical execution plan (LEP) generation, physical execution plan (PEP) generation and PEP execution. LEP and PEP generation and PEP execution are often called the query optimization. The cost models implemented by current query optimizers are very complex and rely on factors which may induce the optimizers to produce inefficient PEPs. In order to overcome such an issue, there are strategies to help optimizers, such as to provide query hints and query rewriting [Garcia-molina et al. 2000].

The rewriting technique consists in writing a new SQL statement Q_b equivalent to the statement Q_a initially formulated. Two queries are equivalent if and only if their execution produces the same result [Ramakrishnan and Gehrke 2002]. Thus, the executions of Q_a and Q_b return the same result. However, Q_b execution provides performance gains w.r.t. Q_a execution. In order to illustrate performance gains achieved by means of query rewriting, consider the queries Q_a and Q_b depicted in Figures 1 and 2, respectively. In order to execute those queries, the TPC-H benchmark's database has been created in a PostgreSQL server.

Looking more closely to Figures 1 and 2, one can observe that Q_a and Q_b are equivalent. Nonetheless, Q_a execution lasts $13.685ms$, while Q_b is executed in $1.825ms$.

Therefore, we may formulate the following hypothesis, the way a query Q is formulated in terms of SQL commands induces the query optimizer to produce a given PEP for Q. To show the veracity of our hypothesis, consider the execution plans for queries Q_a and Q_b depicted in Figures 3 and 4. Those plans have been yielded by PostgreSql's query engine. The PEP showed in Figure 4 is more efficient (1,825 ms) than PEP depicted in Figure 3 (13,685 ms). This fact stems from the least amount of data materialized in the "Materialize" operation, which is performed after the "Aggregate" operation (in the PEP illustrated in Figure 4).

```
SELECT *
FROM orders
WHERE o_totalprice < ALL (SELECT o_totalprice
                          FROM orders
                          WHERE o_orderpriority = '2-HIGH')

Execution time = 13,685 ms
```

Fig. 1. SQL expression for Q_a.

```
SELECT *
FROM orders
WHERE o_totalprice < (SELECT MIN(o_totalprice)
                      FROM orders
                      WHERE o_orderpriority = '2-HIGH')

Execution time = 1,825ms
```

Fig. 2. SQL expression for Q_b.

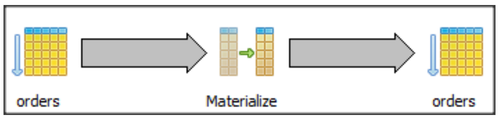

Fig. 3. PEP P_{Q_a} yielded by PostgreSQL for Q_a

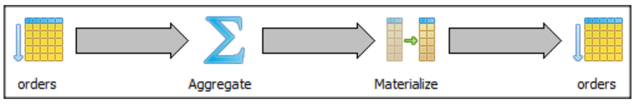

Fig. 4. PEP P_{Q_b} yielded by PostgreSQL for Q_b

Database query tuning explores these aspects in order to help query optimizers to produce better PEPs. In fact, there are tools that aim at tuning database query performance by providing rewriting recommendations. However, they adopt an offline approach, since they provide recommendations only when an human (e.g., DBA) requires such recommendations. Most of them are intrusive, which means that they are DBMS-specific. Additionally, they present a strong issue, for a given query they may provide several rewriting recommendations. In this case, the DBA is in charge to choose one of them without any indication (from the tool) that the chosen recommendation will provide the best performance gains regarding response time.

In this paper, we propose an automatic, autonomous and non-intrusive approach for rewriting SQL queries, denoted ARe-SQL. The proposed approach provides the necessary support to implement two different strategies for SQL-query rewriting. In the first strategy, ARe-SQL autonomously captures SQL queries submitted by users or applications and identify which queries should be rewritten. Thereafter, queries are sent to the database query engine. The second strategy works as an autonomous advisor, which analyze previously executed SQL statements in order to recommend (through alerts, reports and wizards) SQL tuning opportunities. In both strategies, the rewritten queries profit from performance gains as we show in Section 4. We have implemented the proposed approach and evaluated its behavior in three different DBMSs considering the TPC-H benchmark. The results show that our approach can be effectively considered by database professionals.

The rest of the paper is structured as follows. Section 2 discuss most relevant related work. The proposed approach is presented and analyzed in Section 3. In turn, Section 4 brings and analyzes experimental results. Finally, Section 5 concludes this paper.

2. RELATED WORK

Bruno *et al* propose in [Bruno et al. 2009] a framework, called Power Hints, which enables the creation and use of hints. In the proposed framework, hints are created by means of regular expressions, making it easy and flexible to create restrictions for query execution plans, allowing more precise tuning. The tool IBM Optim Development Studio [Studio 2010] collects query performance metrics in DB2. The metrics are query execution frequency, cost and time. Thus, with such metrics, it is possible to identify queries, for which query rewriting recommendation are worthwhile. It is important to mention that this tool collects metrics for a given period of time (defined by the DBA). Embarcadero DB Optimizer XE [Optimizer 2010] has the functionality of identifying hints which should be encoded in a given SQL statement, i.e., it does not propose query rewritings. Its goal is mainly to eliminate unnecessary outer joins and cartesian products. Additionally, Embarcadero may provide recommendation for improving index configuration. Quest SQL Optimizer for Oracle [Quest 2010] provides semantically equivalent SQL statements for a given query. In this case, the DBA should pass to the tool which query should be analyzed. Moreover, it is up to the DBA to choose one of the several recommended SQL statements.

Therefore, all the investigated SQL tuning tools (Automatic SQL Tuning Advisor [Dageville and Dias 2006], IBM Optim Development Studio [Studio 2010], Embarcadero DB Optimizer XE [Optimizer 2010] and Quest SQL Optimizer for Oracle [Quest 2010]) adopt an offline approach. In this sense, they transfer to the DBA the responsibility for defining the set of queries to be evaluated for choosing one of the several alternatives provided by them. Observe that DBAs work in a reactive way, i.e., they only trigger a tool or an advisor when the problem already exists. Furthermore, after identifying the problem (a time consuming query) and a possible solution, a DBA should rewrite the SQL statement, test and send it to a programmer to change the application code which is using the SQL statement. This whole process is rather time consuming.

3. ARE-SQL: AN AUTOMATIC, AUTONOMOUS AND NON-INTRUSIVE APPROACH FOR REWRITING SQL QUERIES

ARe-SQL's main functionality is to influence query optimizer to choose an effectual query execution plan. ARe-SQL proactively rewrites SQL statements which will induce query engine to produce better plans. In order to achieve its goal, ARe-SQL works directed by a set of heuristics. The heuristics consist of rules to identify potential opportunities for tuning SQL statements and the ways to rewrite the statements. Table I brings the eleven heuristics applied by ARe-SQL. Furthermore, it indicates whether or not each heuristic is currently implemented by three major DBMSs: PostgreSQL 8.3, Oracle 11g and SQL Server 2008.

Table I. Heuristics for SQL Tuning.

	Heuristcs for SQL Tuning	PostgreSQL	Oracle	SQL Server
H1	Transform queries which create and use temporary table into an equivalent sub-query.	No	No	No
H2	Eliminate unnecessary GROUP BY.	No	No	No
H3	Remove having clause whose predicates do not have any aggregate function. The predicates should be moved to a WHERE clause.	No	No	No
H4	Change query with disjunction in the WHERE to a union of query results.	No	Yes	No
H5	Remove ALL operation with greater/less-than comparison operators by including a MAX or MIN aggregate function in the subquery.	No	Yes	No
H6	Remove SOME operation with greater/less-than comparison operators by including a MAX or MIN aggregate function in the subquery.	No	Yes	No
H7	Remove ANY operation with greater/less-than comparison operators by including a MAX or MIN aggregate function in the subquery.	No	Yes	No
H8	Replace IN set operation by a join operation.	No	Yes	No
H9	Eliminate unnecessary DISTINCT.	No	No	No
H10	Move function applied to a column index to another position in the expression.	No	No	No
H11	Move arithmetic expression applied to a column index to another position in the expression.	No	No	No

3.1 Implementing ARe-SQL

In order to implement ARe-SQL, two different approaches were employed: assisted and automatic. For each approach, a different tool was implemented. However, both approaches present the following features: *(i)* Non-intrusive: completely decoupled from the source code of the DBMS. This allows that the conceived solution can be used with any DBMS and *(ii)* Independent of location: it can run on a machine different than that used to host the DBMS, not consuming server resources where the DBMS is hosted.

3.1.1 *Assisted Approach.* A query-rewriting tool based on assisted approach consists of an advisor which has the ability: *(i)* to capture the previously SQL statements executed by DBMS; *(ii)* to analyze these statements, and; *(iii)* to recommend (through alerts, reports or wizards) SQL tuning opportunities. Thus, the advisor can identify SQL statements that could be rewritten. Additionally, such a type of tool allow the DBA to interact with the tuning process. For instance, a DBA may select a subset of available heuristics to be applied for the SQL-query tuning process. In other words, the DBA can specify that some heuristics are unnecessary or inappropriate for a given database system or a given statement. We have implemented a tool, called ARe-SQL Advisor, which implements an assisted approach for ARe-SQL. The main components of the architecture illustrated in Figure 5 are the following:

—Agent for Workload Obtainment (AWO): This agent observes the operations submitted to the DBMS and retrieves the SQL statements and then stores them in the local metabase (Local MetaData - LM). This agent can be configured to run continuously (On-the-fly).

—Local Meta Data (LM): Database that stores workloads captured by AWO.

—Driver for Workload Access (DWA): This component enables ARe-SQL Advisor to access metabase (catalog) of a given DBMS.

—Agent for Statistics Obtainment (ASO): This component is in charge of accessing statistics information of the target DBMS, such as table cardinality, the amount of disk pages required to store a database table, the height of (B^+ tree) index structures and so forth.

—Driver for Statistics Access (DSA): driver that allows the ASO to retrieve statistics for a specific DBMS.

—Heuristic Set (HS): set of heuristics used by the agents to identify SQL statements with oportunities of tuning and in order to rewrite them. HS is composed of 11 heuristics, which are depicted in Table I. However, new heuristics can be defined and inserted into HS.

—Agent for SQL Tuning (AST): Component responsible for tuning a particular SQL statement using the set of heuristics (HS).

—Tuning Settings (TS): This component is preference file containing pairs <SQL statement Q, subset of heuristics H> defined by the DBA. Each pair indicates a heuristic subset chosen by DBA from HS that he/she wants to be applied for tuning Q. If there isn't an entry in this file for a given statement Q, all 11 heuristics are applied for tuning Q (in both, Assisted and Automatic Approaches).

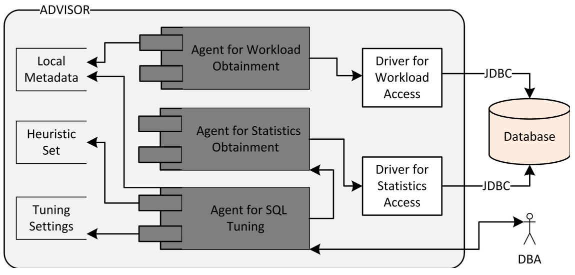
ADVISOR
Local Metadata
Heuristic Set
Tuning Settings
Agent for Workload Obtainment
Agent for Statistics Obtainment
Agent for SQL Tuning
Driver for Workload Access
Driver for Statistics Access
JDBC
JDBC
Database
DBA

Fig. 5. ARe-SQL Advisor's Architecture.

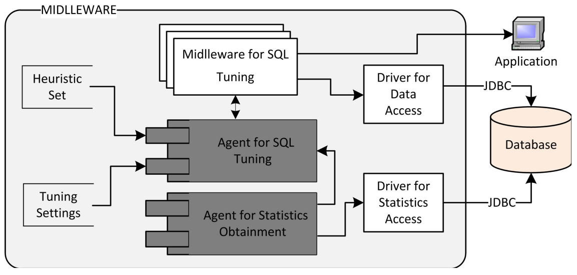

Fig. 6. Architecture of ARe-SQL's Automatic Approach.

3.1.2 *Automatic Approach.* The automatic approach consists of a middleware, ARe-SQL Mid, that works between the application and the DBMS. This middleware is responsible for: *(i)* automatically receiving SQL statements sent by applications; *(ii)* analyzing and rewriting them whenever necessary, and; *(iii)* submitting the statements (rewritten or not) to the DBMS. The architecture designed for the automatic approach is illustrated in Figure 6. The main components of this architecture, in addition to components that are also present in Figure 5, are:

—Middleware for SQL Tuning (MST): is responsible for receiving SQL statements from the applications; send them to the agent AST examine them; and receive the statements rewritten (or not) and send them to the DBMS; receive the result of executing these queries and send them to the applications.

—Driver for Data Access (DDA): driver that allows the engine of the middleware (Middleware for SQL Tuning) to send the SQL statements to the target DBMS.

4. EXPERIMENTAL RESULTS

4.1 Simulation Setup

We have evaluated ARe-SQL in three different scenarios. In the first scenario, TPC-H benchmark has been used, including its database and workload, which is composed of 23 queries. In the second scenario, a workload of 30 synthetic queries has been executed on TPC-H database. In both scenarios, we have used TPC-H benchmark with scale factor of 2 GB. Finally, in the third scenario, we exploit the database of a system used in several brazilian universities, called Integrated Management System (IMS), and another synthetic workload. Each synthetic workload is formed by 30 SQL queries. Each SQL query was written in order to enable the application of one of the heuristics presented in the Table I.

Table II. Execution times of the TPC-H queries 18 and 20 in their original formats and after being rewritten.

	PostgreSQL		SQL Server	
	Original	**Rewritten**	**Original**	**Rewritten**
Query TPC-H 18	134.807ms	111.933ms	19s	14s
Query TPC-H 23	912ms	712ms	15s	11s

All experiments were run on a Core i3-2100 (3.10GHz) server, with 4GB RAM and 500 GB HD. PostgreSQL 8.1, Oracle 11g and SQL Server 2008 have been used as database systems. For each experimentation scenario, three different experiments have been performed: i) The workload containing the original queries has been submitted to a database system. The results of this experiment have been be used as baseline; ii) The workload submitted to ARe-SQL advisor (assisted approach). Thereafter, the workload with tuned SQL queries were manually submitted to a database system, and; iii) Each query from original workload has been sent to the automatic approach of ARe-SQL. For each test we have executed the set of queries belonging to used workload once, twice, 4, 8, 16, and 32 times. For each different number of executions (iterations), the sum of execution time of the whole set of queries was computed.

4.2 Scenario 1: Full TPC-H Benchmark

Figures 7 and 8 bring the result of using the assisted and automatic approach of ARe-SQL running on PostgreSQL and SQL Server, respectively. It is important to observe that from the 23 queries that comprise the TPC-H benchmark, two (18 and 23) have been rewritten by ARe-SQL advisor (see Section 3.1.1). Regarding the results presented in Figures 7 and 8, the assisted approach had a small decrease in workload runtime. This can be explained by the fact that only two TPC-H queries have presented opportunities for tuning. Table II shows the execution time of TPC-H queries 18 and 23 in their original formats and after being rewritten.

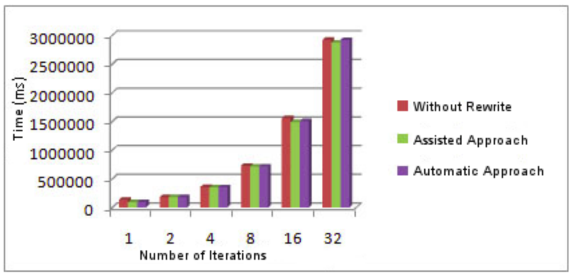

Fig. 7. Benchmark TPC-H on PostgreSQL

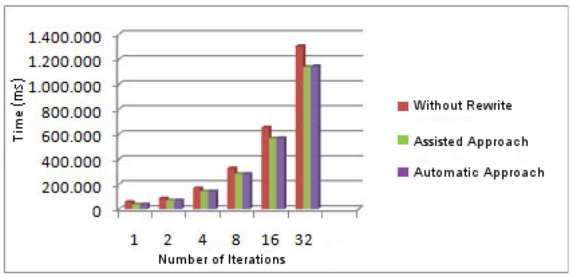

Fig. 8. Benchmark TPC-H on SQL Server

On Oracle, however, TPC-H queries 18 and 23 were not rewritten by ARe-SQL advisor, since Oracle implements heuristic H8. So, the ARe-SQL automatic approach had a slight worsening compared to baseline (24.95%). This is because the automatic approach had the overhead of trying to rewrite both queries.

4.3 Scenario 2: TPC-H Database with Synthetic Workload

Figures 9, 10 and 11 show that using ARe-SQL advisor ensures a significant reduction in query response. On the other hand, ARe-SQL's automatic approach presents smaller benefits than ARe-SQL advisor, since it involves the overhead of tuning the SQL statements received in runtime.

It's important to note that in the test on Oracle (Figure 11) automatic approach presents a small decrease in the execution time of the workload. This is explained by the fact that from the total of eleven heuristics five are already implemented by Oracle (H4, H5, H6, H7 and H8). Besides, from the six remaining heuristics, three of them (H9, H10 and H11) make use of statistical information, and, therefore, require additional access to DBMS, which significantly increased the overhead to rewrite SQL queries.

In the experiment with TPC-H Database and Synthetic Workload on Oracle, the runtime for the 32 iterations of the original workload (without rewriting) took 673.28s, while the runtime for the 32 iterations of the workload generated after applying the assisted approach lasted 49.43 seconds. In this case, assisted approach yielded a gain of 623 seconds (92.5%).

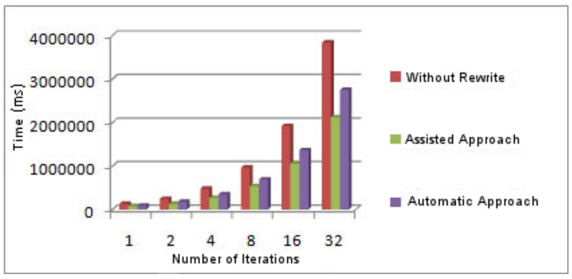

Fig. 9. Synthetic queries in TPC-H database on PostgreSQL

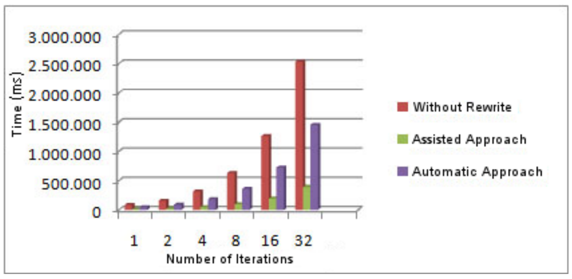

Fig. 10. Synthetic queries in TPC-H database on SQL Server

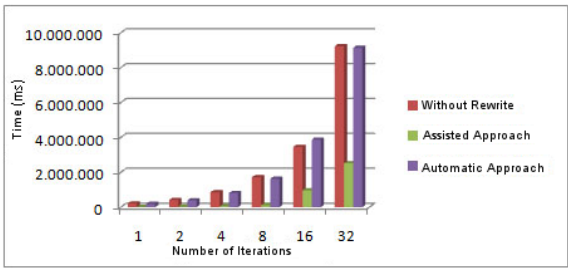

Fig. 11. Synthetic queries in TPC-H database on Oracle

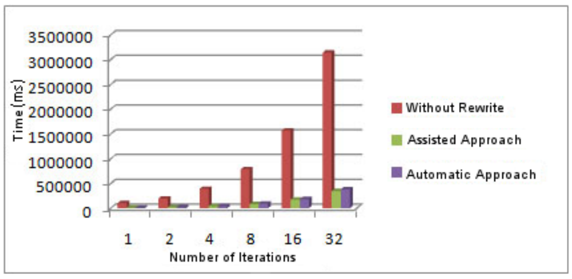

Fig. 12. Synthetic queries in IMS database on PostgreSQL.

4.4 Scenario 3: IMS Database with Synthetic Workload

For this scenario, only PostgreSQL has been used, as the IMS database is only available for that DBMS. Again, the proposed approaches have provided a high reduction in time execution of the submitted workload (Figure 12). In the experiment with IMS Database and Synthetic Workload on PostgreSQL, the runtime for the 32 iterations of the original workload (without rewriting) took 314.32s, while the runtime for the 32 iterations of the workload generated after applying the assisted approach lasted 34.72 seconds. In this case, assisted approach yielded a gain of 280 seconds (89%).

5. CONCLUSIONS

In this paper, we propose an automatic, autonomous and non-intrusive approach for rewriting SQL queries, denoted ARe-SQL. The key goal of ARe-SQL is to assist optimizers in building efficient query physical execution plan. For this, ARe-SQL rewrites SQL statements, using a set of 11 heuristics. Based on ARe-SQL, two different strategies for SQL-query rewriting, denoted assisted and automatic, were implemented. These strategies were evaluated in three different DBMSs considering three different scenarios. The experimental results indicate that both strategies can provide performance gains. In some cases, such performance gains reach 92.5%. The proposed solutions are applicable to situations where the query optimizer cannot produce optimal plans, even using access methods and assessment strategies supported by the DBMS.

REFERENCES

Bruno, N., Chaudhuri, S., and Ramamurthy, R. Power hints for query optimization. In *ICDE '09 Proceedings of the 2009 IEEE International Conference on Data*. IEEE Computer Society, Washington, DC, USA, pp. 469–480, 2009.

Dageville, B. and Dias, K. Oracle's self-tuning architecture and solutions. In *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*. IEEE Computer Society, Oracle, USA, 2006.

Garcia-Molina, H., Ullman, J., and Widom, J. *Database System Implementation*. Prentice Hall, 2000.

Optimizer, E. D. Embarcadero db optimizer xe, 2010. Available: http://www.embarcadero.com/products/db-optimizer-xe. June 2011.

Quest, S. O. f. O. Quest sql optimizer for oracle, 2010. Available: http://www.quest.com/SQL-Optimizer-for-Oracle. June 2011.

Ramakrishnan, R. and Gehrke, J. *Database Management Systems*. McGraw Hill, 2002.

Studio, I. O. D. Ibm optim development studio, 2010. Available: http://www-01.ibm.com/software/data/optim/development-studio. June 2011.