Practical Software Query Optimizing by Adapting Why and How?

Mohammad Reza Feizi Derakhshi, Hasan Asil, Amir Asil, Elnaz Zafarani

Department of Computer, University of Tabriz, Tabriz, Iran
Department of Computer, Islamic Azad University, Tabriz branch (azarShahr), Tabriz, Iran
Department of Computer, University of Tabriz Tabriz, Iran
Department of Computer Engineering, Islamic Azad University, Tabriz Branch, Tabriz, Iran

Abstract: Different methods have been presented because of the need to query processing optimization in databases. The reason of these needs is to increase the amount of data and the queries sent to the Database Management System. Even though there are various methods presented to optimize queries, the problem is that in most of these methods, the execution plan is deleted after the query is conducted. This study tries to provide a method that uses optimized execution plan obtained from executing a query for the next executions or executing a similar query in order to reduce time needed for executing queries. The method presented in this study has been tested on real practical software databases and the results show 11% improvement.

Key words: Relational Database; Agen; Queries Processing; Adaption.

INTRODUCTION

Increasing volume of data and submitted queries to databases in recent years makes DBMS developers to spend more attempt on query optimizing. When Selinger-style query optimization failed, the result (systems flurry and creation of new algorithms) made vendors like Microsoft, IBM and Oracle to investigate and deploy adaptivity features for their database products. Different methods have been presented because of necessity of query processing optimization in databases. The reason of these needs is to increase the amount of data and the queries sent to the Database Management System. Even though there are various methods presented to optimize queries, the problem is that in most of these methods, the execution plan is deleted after the query is conducted. In this paper an attempt is made to use adaptation methods based on numbers and types of submitted queries to practical software database and adapt database as the time goes by. This method increases query-processing optimization.

Aim:

There are various methods presented to optimize query processing in database, but most of these methods use a special procedure (Selinger-style) for optimization and usually they delete plan of query optimization after executing query (Amot Deshpande, 2007) but now a days besides these methods, there are other methods presented which are able to optimize query processing in database. But there is one question: Can we use this optimized plan to execute next queries? In other words cannot we use repetitive queries sent to database for execution of next queries? Of course it’s considered that there are some methods like using cash to do this. In this method optimization occurs by overusing available data in cache (Bingsheng, 2007). As we know in practical software database, sent queries usually own high ability of adaption. On these systems, sent queries to database have the same structure and they will adapt in fastest time.

Literature Review:

Paying attention to the aim of optimization is the best way for enhancer to reach the best optimization techniques, reduce using sources and use the most confident optimization time. Query optimization allows you to reach the goals of optimization. Usually, methods of query processing optimization in database are divided into 3 groups:
A. Server: In this group of method it was tried to reach optimization goals by the changes performed on server like hardware changes.

B. Query: in this group, optimization occurs by some techniques and changes performed on the query. One of these methods is changing parameters of a query.

C. Session: methods of this group do not apply in any of the above groups. Methods like paralleling, use caching and methods in dealing with possibilities place in this group.

In this research trying to present a new method base from the methods pointed out on the third group for optimizing query processing in practical software database.

**Suggested Algorithm:**

This paper suggests a multi agent system for query optimizing. This algorithm attempts to prepare a personalized environment for users and to propose query plan based on users’ queries using data gathering technology.

In practical software databases, queries are submitted to DBMS based on user's needs. Practical softwares usually send queries to database, receive response and reply to the user's needs. In these softwares sent queries were owned by the same structure and they are repeated as time goes by. For this reason we are going to present a method that database identifies queries of the same kind over a period of time. This method also identifies more repetitive queries sent to database and replies to this query by distinct execution plan. In other words we want to adapt database in order to process the queries by prepared execution plans faster and cheaper. In fact we can do this in order to reduce the number of stages that are necessary for query processing for more repetitive queries sent to database.

In this Algorithm an agent adds to DBMS. This agent creates adapted database. This Algorithm tries to increase optimization of query processing in database by reducing necessary stages for processing frequent queries sent to database. This Algorithm consists of the following parts:

A. *Order separator*
B. *Substitution politics*
C. *Queries similarity recognizer*

In this Algorithm we examine the queries first and highlight queries, which own low process cost, or unwanted orders, which are sent to database with low repetition. If orders are unwanted, then it will be executed as a normal query. If they are wanted, then orders are sent to query similarity recognizer part in order to use prepared execution plan to execute the query if it exists among adapted queries. If the query does not exist among adapted queries, it will be executed as a normal query. In addition to this, we need another part that is able to recognize adapted queries and substitute them on the system. We will discuss the details of each part in next paragraph. Figure 1 is a sample of this Algorithm semi code.

**Command Separator:**

Various queries are sent to database and there are some costs to pay on replying in each query. Usually sent orders are divided into 4 groups [4]:

- Select
- Update
- Delete
- INSERT

Each of these queries requires cost to process and various methods have tried to reduce this expenditure. But above all queries there are some exceptions which optimization cannot be performed. There are also some queries, which needs less cost for processing them. Because of that the first aim of our Algorithm is to separate these queries. For example we cannot perform optimization for INSERT orders, because we separate and send them to DBMS as a normal procedure and return a reply to the user after processing. This separator shows in Figure 2. In fact in this Algorithm we are going to separate adaptable queries, exceptions and orders, which we hope, there will be no reduction in cost if we optimize them.
Querries Similarity Recognizer:

For adapting query processing in database, we need a part in suggested system, which able to compare, sent and recognize similar queries. For example notice to below queries:

Select * from tblkala where kalaId=20

Select * from tblkala where kalaId=31

As it is clear these two orders ask details of Kala number 20 and Kala number 31. These queries have the same structure and they can be executed by a plan. This part of system should recognize such orders.

Nowadays, there are various methods presented for optimization, which can select available data in database, by semantic methods and reply to queries in addition to optimization. One of these methods is using semantic caching. This method tries to reply to queries by queries available data in cache and semantic recognition. This method has been presented by canon in April 2007 (Wanhong Xu, 2007). Cache data is examined by semantic recognition of queries and if a part of data exists in cache, then rest of reply will be selected from database by another query. This method makes database reply faster. Figure 3 shows query (Wanhong Xu, 2007).
Fig. 3: Sample of Algorithm (Wanhong Xu, 2007).

Here, method recognizer is used instead of using query data in order to recognize similarity between queries. This method recognize if "we can use a query execution plan to reply another queries or not?" It is considered that all of queries are adapted and they will adapt and compare orderly and with a General Standard.

Replacement Policy:

One of the most important parts of this research are to declare that substitution occurs how, when and by which politics?

As we know adding an agent to database, which always adapts queries, constrains cost to system. For adapting system we examine sent queries to database for a while and the execution plan of similar frequent queries is substituted in database. (It is considered that only queries receipted by separator will be sent to this part). To control increasing volum of database, some of queries should be deleted from database. To achieve this goal, a score are saved for each query. This score will be increased for each submitting of the query. When a query is added to database, a query which has minimum score is deleted from database. Also, all queries are converted to a standard format to decrease comparing time.

System Evaluation:

Nowadays, there are different methods to measure database performance. One of these ways is using execution time on the system. Execution time is needed for sending query and receiving reply.

On Part 4 is a way represented for adapting database. The aim of adapting database is to optimize the query processing. This method is designed and simulated object oriented for evaluation. This system will be added to database as an agent, which is compatible to the 3 sections mentioned in Part 4. Suggested Algorithm is categorized and performed as 4 classes for simulating this system. Then execution results by using this method have been compared with normal state. In addition, we need a DBMS and data according to the dependability of relations. We've used SQL Server as database and SQLToolbelt to create and specify relations of tables. Also, for performing Algorithm and comparing we use VB.Net and API SQL functions. Figure 4 shows the time for executing query (milliseconds).

After simulating system has been done, results are as follows:

1) cost of query execution time normally
2) cost of suggested Algorithm execution
3) cost of adapted query execution as execution plan

To reach results of suggested method we have to add the second and the third cost and compare the result with the first cost.

We have added this agent on application softwares and compare results with ordinary state. On this system the time needed for execution of sent queries to database is compared in 2 state; adapted database and non-adapted database.
Fig. 4: Report from system replying time to adapted queries everyday (sent queries in different days)

Figure 4 shows the rate of necessary time for answering adaptable and non-adaptable queries in each day. This chart shows total required time for executing adaptable queries in non-adaptable state. It was noticed that in this Figure required cost for adapting queries is not added to the above calculations because this system will not be adapted all the time and it will do this only when the traffic is low. In the next evaluations this cost will be spotted.

Figure 5 also shows required time for executing queries everyday in traffic hours of the system. Traffic hour is when many queries are sent to database for execution. We show this figure because we use queries sent to database in traffic hours for adapting.

Fig. 5: Report from system replying time in busy hours everyday

First row in Table 1 shows total time required replying adaptable queries and also reduction of response time for all adapted queries sent to database. Also, this table shows the cost required for adapting queries. On this system, query separator separates some queries when sending queries to database and obstructs their transmission to separator.

The second row shows required time and cost for all of sent queries to database and adapting costs.

<table>
<thead>
<tr>
<th>Table 1: Total system evolution</th>
<th>Decrease response time (%)</th>
<th>Decrease response time</th>
<th>Total Execution time</th>
<th>Type of queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32%</td>
<td>85881.42</td>
<td>268379.41</td>
<td>Adaptable queries</td>
<td>1</td>
</tr>
<tr>
<td>11%</td>
<td>85868.16</td>
<td>780619.633</td>
<td>All queries</td>
<td>2</td>
</tr>
</tbody>
</table>

As it's shown in Table 1, the system response time to queries is reduced up to 11%.

RESULTS AND DISCUSSION

Optimizations have tried to optimize queries based on various methods. But they have deleted query execution plan after optimizing and executing query. Here a method has been presented which is able to increase optimization process in database without changing DBMS structure and by using frequent queries to execute other queries. In this method as it shows in system evaluation part, this adaption has reduced system response time for practical softwares up to 11%. It is expected that by increasing data of practical software’s database rate of this reduction will not change because the required time for reading data is the same in both states. In the future we can adapt queries better and optimize query-processing process more by adding new parameters for adapting.
REFERENCES


Bingsheng He, Qiong Luo, 2007. “Cache-Oblivious Query Processing” Biennial Conference on Innovative Data Systems Research (CIDR) January 7-10, Asilomar, California, USA.

Sybase, Performance and Tuning Series Query Processing and Abstract Plans, Sybase, Inc., One Sybase Drive, Dublin, CA 94568.