A Framework to Subquery Optimization using Case-based Reasoning

Pragya Shukla
Associate Professor
IET, DAVV
Khandwa Road, Indore

Sakshi Mathur
ME Student
IET Girls hostel, DAVV
Khandwa Road, Indore

ABSTRACT
Query optimizers in current database management systems (DBMS) often face problems such as intolerably long optimization time and/or poor optimization results when optimizing complex subqueries using classical techniques [1]. There are computational environments where metadata acquisition and support is very expensive. A ubiquitous computing environment is an appropriate example where classical query optimization techniques are not useful any more. To tackle this challenge, we present a new similarity-based optimization technique using case-based reasoning in this paper[2]. The key idea is to identify cases of similar subqueries that often appear in a complex query and share the optimization result within each case in the query [3]. An efficient algorithm to identify similar queries in a given query and optimize the query based on similarity is presented. Our experimental results demonstrate that the proposed technique is quite promising in optimizing complex subqueries in a DBMS. It is possible to learn from each new experience in order to suggest better solutions to solve future queries.

Keywords
Classical query optimization techniques, ubiquitous computing environment, metadata, case-based reasoning, similarity function

1. INTRODUCTION
Subquery optimization is the process of selecting the most efficient query evaluation plan from among the option available for processing a given subquery. For appropriate performance of query processing it is expected from the system to construct a query evaluation plan that minimizes the cost of query evaluation which can be done through subquery optimizer. Subquery optimizers examine all expression which is equivalent to the new subquery and select the suitable cheapest plan [1]. The area of query optimization in database field is very vast. It has been studied in a great variety of contexts and from many different angles which provides many miscellaneous solutions. Most of these approaches were based on classical subquery optimization, where dependency on metadata is very high. Moreover, classical subquery optimization techniques typically generate query execution plans that are optimized according to a single dimension, query execution time, this characteristic hampers the efficient information access. Due to this dependency it may not work effectively in all type of computational environment. An example of this type of environments is ubiquitous computing environment that integrates information from autonomous, heterogeneous, and dynamic computational tools and applications as well as electronic devices located in a distributed fashion. Ubiquitous environment is one where information technology become pervasive, embedded in environment, heterogeneous, sovereign and invisible to users. According to this vision network will be saturated by computation and wireless communication capacity which will be gracefully integrated with user. Metadata for subquery processing attainment and preservation is not feasible in ubiquitous environment. Thus environment must provide a set of procedures to retrieve information from minimized resources. Additionally, resources used in this environment have physical limitations that restrict their suitable operations like distributed in different locations, limited storage and processing capability, power supply etc. [2].
Here we propose a subquery optimization approach which will deal with these challenges and work effectively in lack of metadata [1], [2].

2. CLASSICAL QUERY OPTIMIZATION
We provide an abstraction of traditional subquery optimization process. The modules that participate in the classical subquery evaluation process are the query parser, query optimizer, code generator and the query executor Fig 1. The query parser is in charge to verify if the query is syntactically (well formed) and semantically correct. The output of this module is a tentative algebraic query tree. It is a sequence of algebraic operations (e.g. selection, projection and joins) that indicate the operations that must be performed on the data for solving the query. Then, a valid query must to be optimized; this is carried out in by the Query optimizer module. That estimates the best order to perform the operations included by the algebraic query tree and assigns to each algebraic operator an algorithm to execute it. The result is the execution plan. When the query is optimized, a codification of the execution tree must be performed by the code generator, to be executed by the last module, the query executor. Finally, the data that solve the query is obtained. Evaluation cost models used for most of classical subquery optimization techniques are tightly tied to metadata use.

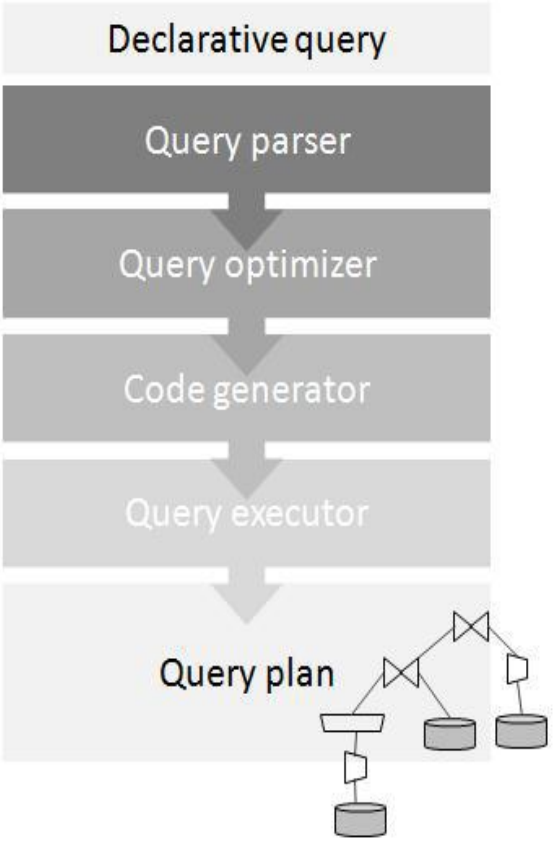

Figure 1: Phases of query optimization

- The Query Parser checks the validity of the query and then translates it into an internal form, usually a relational calculus expression or something equivalent.
- The Query Optimizer examines all algebraic expressions that are equivalent to the given query and chooses the one that is estimated to be the cheapest [6].
- The Code Generator or the Interpreter transforms the access plan generated by the optimizer into calls to the query processor.
- The Query Processor actually executes the query.

Queries are posed to a DBMS by interactive users or by programs written in general-purpose programming languages (e.g., C/C++, Fortran, PL-1) that have queries embedded in them. An interactive (ad hoc) query goes through the entire path shown in Figure 1. On the other hand, an embedded query goes through the first three steps only once, when the program in which it is embedded is compiled (compile time). The code produced by the Code Generator is stored in the database and is simply invoked and executed by the Query Processor whenever control reaches that query during the program execution (run time). Thus, independent of the number of times an embedded query needs to be executed, optimization is not repeated until database updates make the access plan invalid (e.g., index deletion) or highly suboptimal (e.g., extensive changes in database contents) [7].

3. LITERATURE SURVEY

In figure. [3] To a large extent, the success of a DBMS lies in the quality, functionality, and sophistication of its query optimizer, since that determines much of the system's performance. In this paper we have given a bird's eye view of query optimization. We have presented an abstraction of the architecture of a query optimizer and focused on the techniques currently used by most commercial systems for its various modules. In addition, we have provided a glimpse of advanced issues in query optimization, whose solutions have not yet found their way into practical systems, but could certainly do so in the future.

In fig. [2] A wide variety of query optimization techniques as semantic, parametric, and query optimization via probing queries have been suggested. Even though these approaches allow the efficient query processing they are presented in the framework of classical query evaluation procedures that rely upon cost models heavily dependent of metadata (e.g. statistics and cardinality estimates). There exist different computational environments where no metadata are available. This characteristic hampers the efficient information access.

In [1] Case-based reasoning (CBR) is an approach to problem solving that emphasizes the role of prior experience during future problem solving (i.e., new problems are solved by reusing and if necessary adapting the solutions to similar problems that were solved in the past). It has enjoyed considerable success in a wide variety of problem solving tasks and domains. Following a brief overview of the traditional problem solving cycle in CBR, we examine the cognitive science foundations of CBR and its relationship to analogical reasoning. We then review a representative selection of CBR research in the past few decades on aspects of retrieval, reuse, retention.

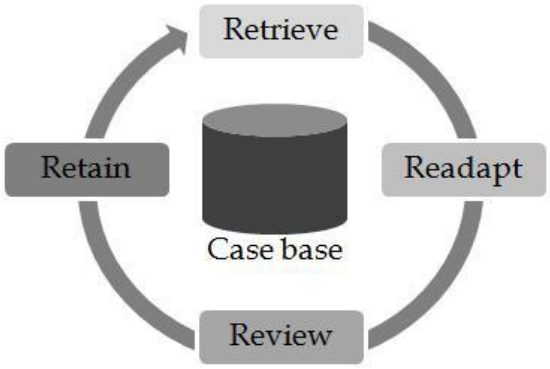

Figure 2: Case based reasoning

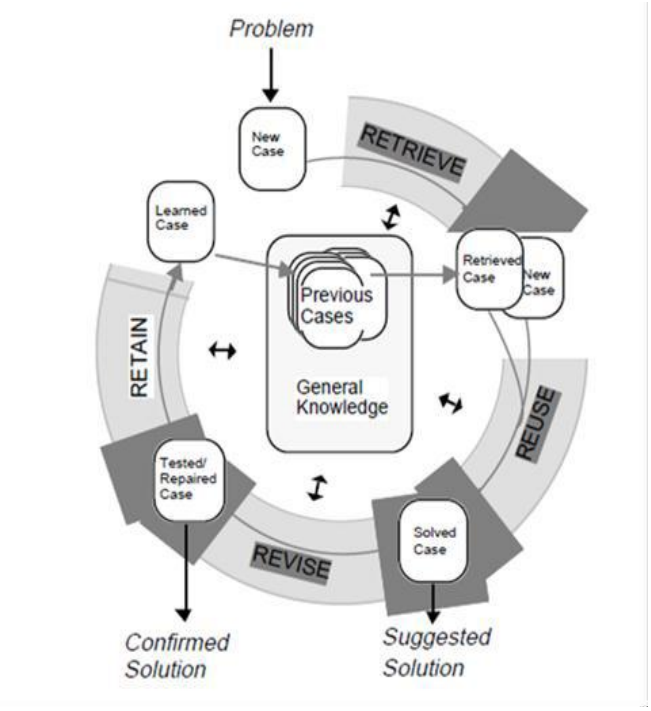

Figure 3: Case based reasoning cycle

4. CASE BASED REASONING APPROACH

Unlike most problem solving methodologies in artificial intelligence (AI), CBR is memory based, thus reflecting human use of remembered problems and solutions as starting point for new problem solving. An observation on which problem solving is based in CBR, namely that similar problems have similar solutions, has been shown to hold in expectation for simple scenarios and is empirically validated in many real-world domains. Solving a problem by CBR involves obtaining a problem description, measuring the similarity of the current problem to previous problems stored in a case base (or memory) with their known solutions, retrieving one or more similar cases and attempting to reuse the solution of one of the retrieved cases, possibly after adapting it to account for differences in problem descriptions. The solution proposed by the system is then evaluated (e.g., by being applied to the initial problem or assessed by a domain expert). Following revision of the proposed solution if required in light of its evaluation, the problem description and its solution can then be retained as a new case, and the system has learned to solve a new problem.
The reasoning process that must be accomplished to optimize a query problem is elaborate in the following

Retrieval: This step involves retrieving of stored cases from case base which is relevant to a new query. Retrieval step is based on a similarity function in order to perform a smart search to retrieve the most relevant case to solve the query problem. Among these relevant cases, the one that minimizes the cost function of the problem is selected.

Reuse: Reusing is followed by retrieval step, it is also known as readaptation. Reusing step is related to the adaptation process of the execution plan involved by the case that resulted relevant to solve the new query.

Review: Reviewing step consists in verify the query by means of its execution. During this step measures about performance as well as computational resources consumption are taken.

Retention: Finally, in the retaining step, approach to incremental, sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future problems the problem and its solution are stored in the case base in form of a new case.

Since this approach is based in a try and learn principle, when a relevant case to solve a query problem is not founded in the case base, is necessary to propose a new solution [5].

4.1 Similarity Function

Similarity notion is useful in different steps of the case based reasoning process. At the retrieving step, a relevant case is identified by applying a similarity function. Furthermore, at adaptation step, the matching process depends on how much similar is the relevant case to the query problem. Finally, at the retaining step, a case is stored in the base of cases according to a defined classification. It is possible to know to which class pertains a case determining the similarity between the class of the query problem and the class in the case base. The formalization of the original definition is expressed as follows:

$$S(a, b) = \theta f(A \cap B) - \alpha f(A - B) - \beta f(B - A)$$

Similarity between a and b, is defined in terms of the features common to a and b, $A \cap B$, the features that pertain to a but no to b, $A - B$, and those that pertain to b but no to a, $B - A$. The variables θ, α, and β are non-negative valued free parameters that determine the relative weight of these three components of similarity. Such variables provide the flexibility when modifying the importance of similarities or differences that in conjunction determine the similarity between two elements according to the area of application. The function f measures the silence of a particular set of features (also can be a single feature)[8].

4.1.1 Inter-class similarity: Inter-class similarity is defined as an increasing function of common operation families and as a decreasing function of distinctive families, in other words, families that pertain to one query but not the other [9]. The formalization of this definition in terms of the similarity between a query and a class is expressed as follows:

$$S(C1, Q) = \theta\_(C1 \cap Q) - \alpha\_(C1-Q) - \beta\_(Q-C1)$$

Similarity between C1 and Q, is defined in terms of operation families common to C1 and Q, $C1 \cap Q$, the features that pertain to C1 but no to Q, $C1 - Q$, and those that pertain to Q but no to C1, $Q - C1$. The function f refers particularly to operation families _. According to the purpose of our work, these are the features that must be compared.

4.1.2 Intra-class similarity: Intra-class similarity function aims to find the most similar queries with respect to a new query, which is desired to be optimized, within the same class [9]. The formalization of this definition is as follows:

$$S(Q1, Q2) = \theta o(Q1 \cap Q2) - \alpha o(Q1 - Q2) - \beta o(Q1 - Q2)$$

Similarity between Q1 and Q2 is defined in terms of the operations that are common to Q1 and the features that pertain

to Q1 but no to Q2. It is possible to establish a mapping one to one for each of the operations included in Q1 and Q3 according to the comparison operator that each of them applies. Q1 and Q2 have two operations in common; they differ in the operator applied by the join operation. According to this simple analysis, Q3 is the most similar query with respect to Q1 because it contains the maximum number of operation mappings.

5. SUBQUERY OPTIMIZATION USING CASE BASED REASONING

The subquery optimization technique that we propose is an adaptation of the general case-based reasoning process. This technique aims to solve the problem of lack of metadata, thus it is feasible to be applied in different execution environments which can't afford expensive acquisition and maintenance of metadata [11]. Since case and problem are the main units of knowledge in this learning approach, we select useful knowledge for query optimization in order to instantiate both concepts. According to our approach, a case represents the knowledge related to the experience gained from the optimization and evaluation of a subquery. A problem represents a new query, that we call query problem, which is submitted in some application pertaining to the ubiquitous environment. We propose subquery optimization strategy that adapts case based reasoning in order to provide optimal execution plans to solve new queries. This strategy recovers, adapts or generates execution plans using the knowledge acquired from the experience to optimize and execute similar subquery.

5.1. Algorithm

Step 1: Receive new query from user, extract subquery (Q) from it.

Step 2: Generate a case on the basis of subquery (in the form of from, where clause)in the given query.

Step 3: Retrieval process initiated

Searching for the appropriate stored case (i.e. in form of subquery) using inter class similarity function.

Comparing for most similar subquery with respect to new subquery using intra class similarity function

On basis of above functions

If (similar case found for Q)

Retrieval process completed (output relevant stored case(C) for new query (Q))

Else

{

New query execution plan will be generated for Q.

Adding a new plan (in form of case) to case base and exit.

}

Step 4: Readapting stored case C (output of retrieval process) according to new query Q.

Step 5: Review of result (execution of query accordingly and output is delivered).

Step 6: Retention of case in case base for future use (on basis of case replacement policy).

5.2. Example

```
Query Q
Select o_orderpriority, COUNT(*)
From orders as o
Where o_orderno > 12 AND
Exists( Select *
        From lineitem as l
        Where o_orderkey =  l_orderkey,
              l_returnflag = 'R' );
```

The query Q contain a subquery having three clauses select from and where. As it will be the new query it will be stored in the case base in the form of a new learned case. As it is stored in the case base as learned case now for any new query which is similar to this query, we can reuse it from the case base and readapt the values according to the new query.

```
Query Q'
Select o_orderpriority, COUNT(*)
From orders as o
Where o_orderno > 12 AND
Exists( Select *
        From lineitem as l
        Where o_orderkey =  l_orderkey,
              l_recietdate > l_commitdate );
```

With reference of above example in Q and Q' first comparison will be done through inter class similarity function which will be performed on the basis of main class category (From clause). Here in this example we can see that Q and Q' were belonging to the same main class thus second step of similarity check will be perform i.e. is intra class similarity check to find out the most relevant case. Now comparison will be done on basis of sub classes based on where clause.

WHERE clause of Q = o_orderkey = l_orderkey, l_returnflag='R'

WHERE clause of Q' = o_orderkey = l_orderkey, l_recietdate>l_commitdate

As both queries contains different where clause condition values but Q can be considered as most relevant case for Q'. After completion of the retrieval process next step comes is reuse where we have to adapt our retrieval case according to a new query problem using similarity level [10]. The similarity level between two queries indicates which clauses of the relevant query must be adapted. This adaptation can be performed only on Select and Where clauses. Reason behind this is that for Select clause, interesting attributes to be projected can vary and for where clause, comparison operators or some values related to the variables can be modified. On the other hand, the from clause cannot be changed because the tables to be queried cannot be changed.

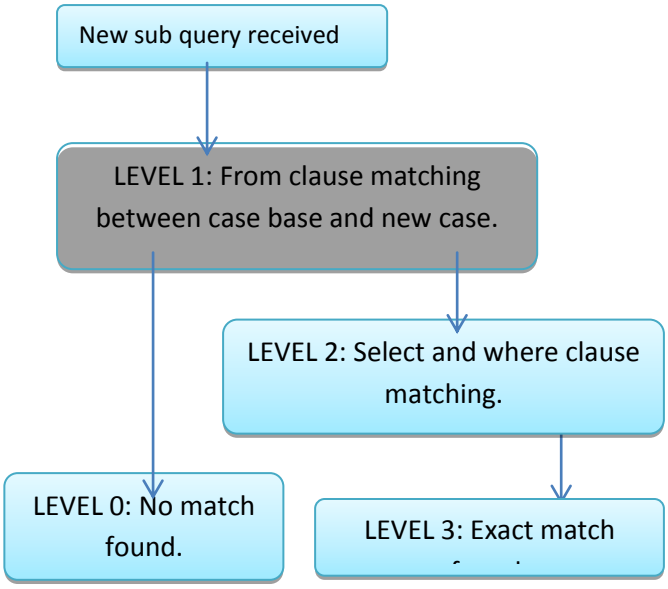

Figure 4: Flow chart for similarity level

In our example similarity level between both the queries Q and Q' is 2, modification will performed only in where clause value which is

WHERE clause of Q = o_orderkey = l_orderkey, l_returnflag='R'

WHERE clause of Q" = o_orderkey = l_orderkey, l_recietdate>l_commitdate

Next is review step where proposed solution is verified through execution. Finally in the end retention of newly learned case is performed for future use. At the retaining step, a case is stored in case base, according to a defined classification. It is possible to know to which class pertains a case determining the similarity between the class of the query problem and the class in the case base.

6. FUTURE WORK

We propose a new query optimization technique that exploits case-based reasoning in order to improve query optimization in ubiquitous environments [3] [4]. Our approach deals with the challenge that the lack of metadata implies in this execution context. We propose a technique that is based on the useful knowledge (resource consumption measures) obtained from previous query executions [7]. In addition, we propose a technique that allows the configuration of the optimization objective according to the users and application requirements, even for each single query. The most important contributions of our work are centered in the reasoning steps related to retrieval and re-adaptation of the useful knowledge. These steps retrieve the most relevant cases and adapt a previous solution to the new situation.

Currently we are working on a prototype which only runs on SQL but no other framework, so related work can be done to make it platform independent. Dynamicity management is also an important point to work on. However, the system should be able to detect those cases in its case base no longer relevant and thus delete them. This is a problem of knowledge

7. REFERENCES

[1] Lourdes Ang´elica Mart´ınez-Medina and Christophe Bibineau and Jose Luis Zechinelli-Martini, Query optimization using case-based reasoning in ubiquitous environments in Mexican International Conference on Computer Science, 2009.

[2] Silberschatz- Korth- Sudarshan, "Database System Concepts", Fourth Edition copyright © by Foxit Software Company, 2004

[3] Syedur Rahman1 , A. M. Ahsan Feroz2, Md. Kamruzzaman3 and Meherun Nesa Faruque4," Analyze Database Optimization Techniques", IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.8, August 2010.

[4] L. D. Mantaras, R. McSherry, and et al, "Retrieval, reuse, revision and retention in case-based reasoning," Knowl. Eng. Rev., vol. 20, no. 3, pp. 215–240, 2005.

[5] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," AI Communications, vol. 7, no. 1, pp. 39–59, 1994.

[6] Y. Ioannidis, "Query optimization," ACM Comput. Surv., vol. 28, no. 1, pp. 121–123, 1996..

[7] M. Franklin, "Challenges in ubiquitous data management," in Informatics - 10 Years Back. 10 Years Ahead., R. Wilhelm, Ed. Springer-Verlag, 2001, pp. 24–33.

[8] M. Gu, X. Tong, and A. Aamodt, "Comparing similarity calculation methods in conversational cbr," in In: Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration, 2005, pp. 427–432.

[9] A. Tversky and I. Gati, "Studies of similarity," 1978.

[10] R. Bergmann and A. Stahl, "Similarity measures for object oriented case representations," in In: Proceedings of the 4th European Workshop on Advances in Case-Based Reasoning B, B. Smyth and P. Cunningham, Eds. Springer Verlag, 1998.

[11] Christiane Gresse von Wangenheim, "Case Based Reasoning- A Short Introduction" in University of Italy in 2000