QUERY OPTIMIZATION IN OODBMS: IDENTIFYING SUBQUERY FOR QUERY MANAGEMENT

Sheetal S. Dhande[1] & Dr. G. R. Bamnote[2]

[1]Department of Computer Engineering, Sipna's College of Engg & Tech, Amravati, Maharashtra, India
[2]Prof & HOD Computer Science &Engineering Prof. Ram Meghe. Institute of Research, Badnera Amravati, Maharashtra, India

ABSTRACT

This paper is based on relatively newer approach for query optimization in object databases, which uses query decomposition and cached query results to improve execution a query. Issues that are focused here is fast retrieval and high reuse of cached queries, Decompose Query into Sub query, Decomposition of complex queries into smaller for fast retrieval of result.

Here we try to address another open area of query caching like handling wider queries. By using some parts of cached results helpful for answering other queries (wider Queries) and combining many cached queries while producing the result.

Multiple experiments were performed to prove the productivity of this newer way of optimizing a query. The limitation of this technique is that it's useful especially in scenarios where data manipulation rate is very low as compared to data retrieval rate.

KEYWORDS

Query Caching, Query Decomposition, Query Optimization, Stack Based Approach (SBA), Stack-Based Query Language (SBQL), Object Databases.

1. INTRODUCTION

Object-Oriented database technology is a marriage of object-oriented programming and database technologies. How these programming and database concepts have come together to provide what we now call object-oriented databases. Perhaps the most significant characteristic of object-oriented database technology is that it combines object-oriented programming with database technology to provide an integrated application development system. There are many advantages, including the definition of operations with the definition of data. First, the defined operations apply ubiquitously and are not dependent on the particular database application running at the moment. Second, the data types can be extended to support complex data such as multi-media by defining new object classes that have operations to support the new kind of information.

Query processing and its optimization are the two most popular areas of research in the database community. Query processing is the sequence of actions that takes as input a query formulated in the user language and delivers as result the data asked for [1]. Query processing involves query transformation and query execution. Query transformation is the mapping of queries and query results back and forth through the different levels of the DBMS. Query execution is the actual data retrieval according to some access plan, i.e. a sequence of operations in the physical access language [2].

DBMSs support high performance of query processing by using special data structures (e. g. indexes, stacks) or by performing operations like I/O operation simultaneously. However, the major potential of performance improvement can be achieved by special preprocessing of query before execution .This preprocessing referred to as query optimization, is performed by special module called query optimizer.[3]

The task of query optimization ideally is to find the best execution plan, i.e. the execution plan that costs the least, according to some performance measure. Usually, one has to accept just feasible execution plans, because the number of semantically equivalent plans is too large to allow for enumerative search [4].

In various types of Database Systems (Relational as well as Object-Oriented), many techniques for query optimization are available [5]. Few of them are Pipelining, Parallel Execution, Partitioning, Indexes, Materialized Views, and Hints etc [6] [7].

One technique which has not been convincingly implemented is Query Caching [8].

Query Caching will provide optimum performance. Instead of spending time re-evaluating the query, the database can directly fetch the results from already stored cache. The most obvious benefit of Query Caching can be seen in systems where Data Retrieval rate is very high when compared to Data Manipulation. Hence database i.e. data store get modified after the long periodic intervals. During these intervals if a particular [25]

2. RELATED WORK

The task of a Database Management System (DBMS) is to safely store usually large amounts of consistent data and to provide easy and fast access to these data, either for retrieval or update purposes. The data model of a DBMS provides possible structure of the data so that it provides easy access to the user. The implementation of such a high-level query language requires an enormous effort; it is the task of the query optimizer to ensure fast access to the data stored in the database [1].

A DBMS is a complex piece of software that consists of many layers. The three main layers distinguished are the user interface, an intermediate layer that is called the logical algebra, and the bottom layer called the physical algebra, which provides mechanisms to actually access the data stored in files.

[5][9]In query processing, the user query is first mapped into a logical algebra expression, and this expression in turn is mapped into an expression of the physical algebra. Traditionally, query optimization consists of two phases:

- Logical optimization, which is the rewriting of a logical algebra expression into one that (hopefully), can be evaluated with less cost.
- Cost-based optimization, which concerns the mapping of logical algebra expressions into expressions of the physical algebra.

Object identity can be employed to speed up join processing, the presence of inheritance hierarchies and path expressions allows to design new index structures.

New approach present in [8]for coupling OODBMSs (Object Oriented Database Management Systems) and IRSs (Information Retrieval Systems) that provides enhanced flexibility and functionality ,This approach allows to decide freely to which document collections, that are used as retrieval context, document objects belong, which text contents they provide for retrieval and how they derive their associated retrieval values, either directly from the retrieval machine or from the values of related objects

Seen the success [8][9]that the query optimization knew in the relational model, in OODBMS data are represented in the basis as of objects. Associations are implemented by the direct ties via object identifying that permit a fast navigational access between the different objects.

[11]The usual cost measure especially for large databases is the total no of disk access (since storage access is most costly operation) such number is determined among others by the size of page which is an atomic unit of data that can be processed by a system (the most popular size of page is 4-8 KB).For smaller databases, where most of the data involved in a query can be completely stored in memory, the emphasis is usually on minimizing computation cost.

[12]While specifying in the class diagram a mono-valuated or multi-valuated tie toward another class, queries can follow this ties in order to attend the searched objects. Specific features of object oriented data models offer many additional opportunities for optimization.

Cost-based optimization [25] [26] is guided by specific database characteristics such as table size (i.e. cardinality and tuple width), the presence of indices, etc. Usually, user languages are high-level, declarative languages allowing stating what data should be retrieved, not how to retrieve them. For each user query, many different execution plans exist, each having its own associated costs. The task of query optimization ideally is to find the best execution plan, i.e. the execution plan that costs the least, according to some performance measure.

From the conceptual point of [13][14] view transparency is the most essential property of a cached query. Caching of query results yields relatively most improvement in a query evaluation performance, i.e. significantly decreases the time of anticipation for a response from database management system. Additional advantage of the query caching method is that reuse time of cached results is independent of query type, its complexity and current database state. The optimization method needs some time for storing in the cache queries, their results together with proper structures for maintenance purposes.
[15] [16]. Database operations typically involve obtaining a database root from the OODBMS which is usually a data structure like a graph, vector, hash table, or set and traversing it to obtain objects to create, update or delete from the database. When a client requests an object from the database, the object is transferred from the database into the application's cache where it can be used either as a transient value that is disconnected from its representation in the database

(updates to the cached object do not affect the object in the database) or it can be used as a mirror of the version in the database in that updates to the object are reflected in the database and changes to object in the database require that the object is refetched from the OODBMS.

[17] Accessing secondary storage is much more expensive than accessing main memory, therefore some data (partial query result) are stored in special main memory buffers.

[18] sometime function can be very expensive to execute, it is sometimes advantageous to cache its result in case it is invoked multiple times with the same arguments. In such cases function input and output are stored in a special data structure. If the function is invoked afterwards for same arguments, it is enough to find the appropriate value in the structure , the subprogram does not have to be executed again . there are three main methods to implements function caching.

Memorization: computation result are stored in main memory hash table.

Sorting: it is useful when an expensive function is to be executed for the value of relation column; the input relation is stored on the input column and cache of only the last input/output pair for the function is maintained

Hybrid cache: the basic idea is to be memorization , but to manage the input stream so that main memory hash table never exceeds a maximum size; this is accomplished by staging tuples with previously unseen input value to disk and rescanning them later

Query is calculated only once i.e. for the first time and 999 times the stored result is reused [15]. Data Manipulation can invalidate the cache results because the inserted /modified /deleted data can bring the difference between the cached results and the actual results. Hence regeneration of the cached results will be required to restore the results back again to the useful state .

Oracle 11g database system offers this mechanism for SQL and PLSQL. Mysql database also implemented query chaining where only full selected query texts together with the corresponding result stored in cache. LINQ, Microsoft .NET environment also have this kind of facility [5][19].

Our research is focused on how cached queries transparent mechanism can be used in query optimization, assuming no changes to syntax, semantics and pragmatics of query language itself [16]. But there is not any result caching solution implemented in current commercial and non – commercial object oriented database system [20].

The functionality of a currently development database programming methodology called ODRA (Object Database for Rapid Application development) which works fully on the object oriented principles. The database programming language is called SBQL (Stack based query language).There are different optimization techniques, which are available in ODRA like [21]

- Query Optimization through Cached Queries for Object- Oriented Query Language SBQL.
- Query Optimization by Result Caching in the Stack-Based Approach.
- Query Optimization by Rewriting Compound Weakly Dependent Subqueries.

The most important concept of this work, Query caching have been implemented by constructing a prototype [22] which suggested in future scope (methods for reusing of some parts of cached results or results of many cached queries combined together.) in Query Optimization through Cached Queries for Object- Oriented Query Language SBQL

The experimentation for optimization of query will be done based on queries written in Stack based query language (SBQL) syntax which is designed and implemented using a stack-based architecture (SBA) framework [23][24].The performance gains will be measured by comparing the results against the performance of the identical queries written and executed in Prototype with cached and without cached queries.

This paper organized as section 1 Introduction. Section 2 Related work about Query optimization in object database. Sections 3 stack base Approach and section 4 describes description of optimization strategies-Query caching and further reuse the result in cached by earlier execution. Section 5 shows experimental findings and section 6 Discuss Experimental Result of cached semantically mapped queries and concludes.

3. STACK-BASED APPROACH

Stack-Based Query Language (SBQL) is useful for the design and implementation of wide range of database models [23]. SBQL is developed according to the Stack-Based Architecture (SBA), a conceptual framework for developing object-oriented query and programming languages [24]. In SBQL the data is stored in the form of persistent objects and the collection of data objects is called as Object Store. Hence adding, deleting or updating information in Object-oriented Databases is nothing but adding, deleting or updating the objects. Objects may contain other objects (aggregation) or references to other objects. Hence the Object-Oriented Modeling concepts of complex objects, associations between objects, classes, types, methods, inheritance, dynamic roles, encapsulation, polymorphism, semi-structured data and other features are employed in the creation of Object Store Models, a representation of the database in Object Databases.

3.1 Query Language (SBQL)

SBQL permits the use of all well-known query operators such as selection, projection, navigation, path expressions, join, quantifiers, etc. SBQL has special as and group as alias operators, apart from binary operators [either of algebraic or non-algebraic type.

In the evaluation of SBQL queries two stack are in use namely ENVS (Environmental Stack) and QRES (Result Stack). In the processing of algebraic operators ENVS is not required to be used [24]. The examples of algebraic operators are Operators for arithmetic and string comparisons, set-oriented operators, aggregate functions, auxiliary naming operators, Boolean operators, etc. In contrast, processing non-algebraic operators involves operations on ENVS. The examples of non-algebraic operators are selection (where), projection/navigation and path expressions (dot), dependent join (join) etc.

3.1.1 Distinction of SBQL Queries

In contrast to SQL and OQL, SBQL queries can be easily decomposed into subqueries, down to atomic ones, connected by unary or binary operators. This property simplifies implementation. In query optimization decomposed atomic queries along with query caching plays an important role.

3.2 Object Database Models

SBA assumes a family of object store models which are enumerated AS0, AS1, AS2 and AS31. The simplest is AS0, which covers relational, nested-relational and XML-oriented databases. AS0 assumes hierarchical objects with no limitations concerning the nesting of objects and collections. AS0 also covers pointer links (relationships) between objects.

4. QUERY OPTIMIZATION ARCHITECTURE

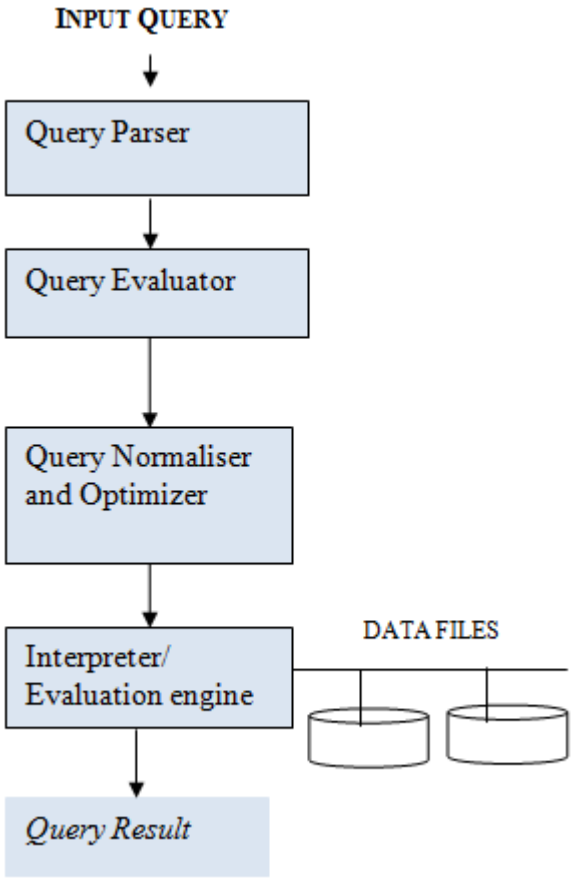
Trasform query into syntatic tree
.

Static evaluator and type checker , whether query
syntatically correct , validate table name , coloum name,
operator , object name , method name , function name involed in query.

According to rules of normalization and send query
to query optimizer (which is based on chche query
optimizer)

Interpret query , find result.

Find and dispaly query result.



4.1 Query Optimization

The scenario of the optimization using cached queries in query evaluation environment for SBA is as follows [10].

1) A user submits a query to a client-side user interface.
2) The user interface system passes it to the parser. The parser receives it and transforms into a syntactic tree
3) The query syntax tree is then received by static evaluator and type checker. It checks whether the query is syntactically correct or not. If not, it will report the errors. It also validates the table names, column names, operators, procedure names, function parameters involved query. Hence it will check the query's semantics. For this purpose, it will use the Metabase present on the server side. Metabase is a part of the database system which contains the Meta information related with the data in the various objects. This is static evaluation of the various nodes in the query syntax tree [12].
4) This type checked and statically evaluated query tree is then sent to the query normaliser which reconstructs the query according to the rules of normalization. This normalised query is then send to the query optimizer. All these components query parser, query type checker; query normaliser, query optimizer and query interpreter are employed on the client side database system.
5) The query optimizer rewrites the received normalized query using particular strategies like query decomposition. Each decomposed part of the complex query is send to the server [15]. Server checks whether the received sub query is already cached or not. If sub query is present in cache, the Unique Identification number of the entry in cache which corresponds to the result of the given sub query is dispatched to the query optimizer. Optimizer replaces (rewrites) the sub query tree of the total query tree by a node containing that unique identification number [16]. This UIN will be used by query interpreter to directly fetch the result from the server. Hence all the parts of the query whose results are already stored in cache will get replaced by their respective UIN. Due to this all sub queries which get replaced by corresponding UIN, their results will be brought from the cache & hence their re-evaluation will be avoided. This rewriting will generate the best evaluation plan which promises to give the best performance & having a least cost in terms of time and storage.
6) The optimized query evaluation plan is then sent to query interpreter.
7) The plan is executed by the query interpreter.

4.2 Class Diagram of Example Database

To implement this prototype we have design Match database. The additional information included is as follows

- Information about officials involved in a test match will be included in the database. These people include umpire, TV umpire, match referee and reserve umpire.
- To illustrate inheritance, here we created class *person*. All people participating in test match will inherit properties from object *person*.

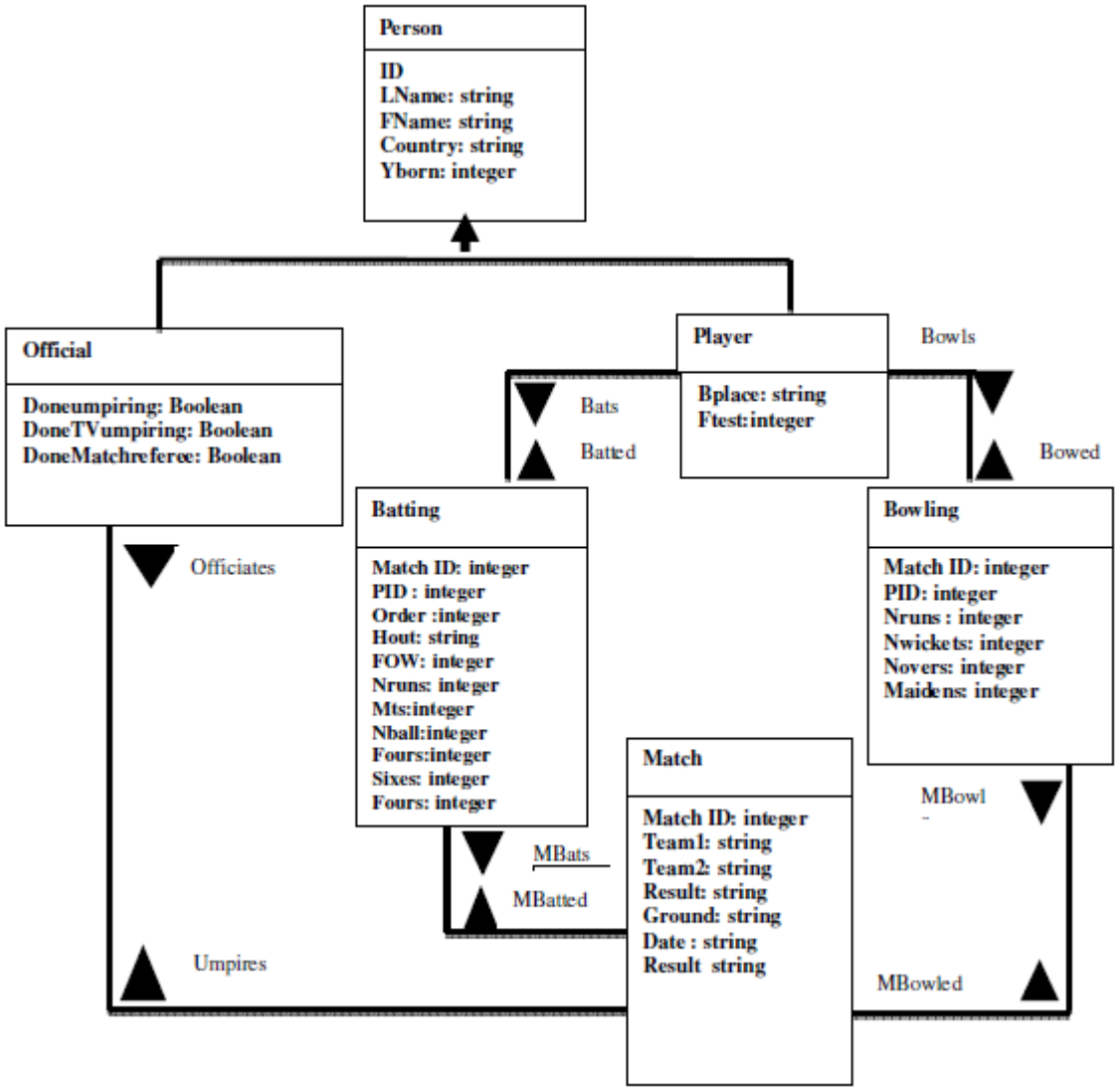


Figure 2: Class Diagram of Match Database

4.3 Query Caching

Once the optimised evaluation plan is executed successfully, the query is cached on the server side in pair <query, result>. Following that the calculated result of the query is send to the client user who has submitted the query.

When the semantically equivalent query (written in the same or different way) is submitted by the same or other user, after parsing, type checking and normalization of the query, optimizer sends the query to the server side query cache manager. Query cache manager searches for the query in the query cache registry and if found there will return the unique identification number (UIN) of the corresponding result to the client, thus avoiding the recalculation of previously stored result. Using this UIN, query interpreter (on the client system) can fetch the stored result of the query directly from the server. If the query is not found in query cache registry, query cache manager

will send a message to an optimizer (on the client side database system) indicating that a query is not cached and hence its result needs to be calculated. Optimizer then does not rewrite the query i.e. does not reconstruct a parse tree. That part of the query will be then calculated by the query interpreter at runtime using runtime ENVS (Environmental Stack) and runtime QRES (Result Stack) [22] [24].

Description of few components on server side:

Query Cache Manager – This is a program running in the server, its job is to check the Query Cache Registry and figure out if the query is cached. The Query Optimizer with pass normalized query (or normalised inner sub-query) to the Query Cache Manager.

Query Cache Registry – This contains all the cached queries along with the results.

4.4 Query Normalization

To prevent from placing in the cache, queries with different textual forms but the same semantic meaning (& hence also will generate the same result), several query text normalization methods will be used. Hence if a query is already stored in the cached with its result, all semantically equivalent queries will make reuse of the stored result, as all those queries will be mapped with the already stored query (due to normalization) [9].

Examples of few techniques useful in the process of normalization are:

a) Ordering of operands

Sum and multiply operations are put before subtractions or divisions [23], i.e. an arithmetic expression is transformed as follows:

p / q / r * s / t

is normalized to:

p * s / q / r / t

b) Unification of Auxiliary Names

Auxiliary name used here for *as* or *group as* operators are unified. , it is not root of syntactic tree, in other case , it ii memorized query result, evaluated earlier by static evaluator.

Query Q*:*
((fname where country = “india”) as f) join (f.batting as h)) . f . Ftest, h.MatchId)

Is normalized to

(((fname where country = “india”) as cache _aux1) join (cache _aux1.batting as cache_aux2)) . cache _aux1. Ftest, cache_aux2.MatchId)

c) Ordering based on column names (in the order in which they appear in the object description).
d) Column names should be maintained to the left side of each operator .

4.5 Query Decomposition and Rewriting

After normalization phase query is decomposed, if possible, into one or many simpler candidate sub queries. Query decomposition is a useful mechanism to speed up evaluating a greater number of new queries. If we materialize a small independent subquery instead of a whole complex query, then the probability of reusing of its results is risen. In addition, a simple semantic of the decomposed query reduces the costs of its updating. Each isolated subquery and finally a whole query is independently analyzed in context of the set of cached queries defined in the query cache registry and if it hasn't yet cached, it becomes a new candidate for caching.

Decomposition of queries in our proposed approached based on operators in SBQL , that are algebraic and non algebraic in categories.

Nature of operator decided query is dependent and independent, if query join with algebraic operator , always consider independent subqueries . if query involves a non algebraic operator , then subquery evaluated in the context determined by another subquery involved query.

4.5.1 Operator that identifies subqueries

In SBQL queries are combined by operators , which is algebraic and non algebraic , the majority of operators in sbql are algebraic . they include numerical comparisons , numerical operators , string comparisons, Boolean and, or, not , aggregate function , set ,bag sequence operators and comparisons , the Cartesian product (denoted by comma), etc[24].

Algebraic Operator

The definitions of algebraic operator are follows

Let q1 Δ q2 be a query formed of two binary algebraic operator Δ, evaluation of this operation is as follows

Algo *evaluatequery(query)*

Begin

……

……

resultarray : array

*Case query is q1 Δ q2 (* Δ is algebraic operator)*

Begin

Evaluate query q1 and push the result into result stack

Evaluate query q2 and push the result into result stack

Read the result of q2

Save the result of q2 in resultarray

Read the result q1

Save the result of q1 in resultarray+1

Apply Δ (temp Δ temp+1) to the result of both the subqueries and save the final result in result stack

End.

And another useful algebraic operators is defining of an auxiliary name n (for the result of query q). it assigns the name n to each row of the result table / result array returned by q

Following are semantics

Match

Return the result table/ result array

<i1> , <i2>

Then the query

Match as *M*

Return the result table/ result array

{<M(i1)>, <M(i2)>}

Group as

This names entire result of query. The semantics of this operator is as follows: if q returns a resulttable /result array t , then query

q **group as** N

returns a result of N(t) .

Non algebraic operator

The evaluation of non algebraic operators (a selection , a dot, a dependent join, quantifiers, etc) is more complicated. In algebraic operator, subqueries q1 and q2 occurring in query q1 Δ q2 are evaluated independently, then result make up the final query result. In non algebraic operator if query q1 and query q2 involves non algebraic operator θ, then q2 is evaluated in the context determined by q1. This is the reason this operator are referred as non algebraic.

Algo eval(query);

Begin

.....

Case query is q1 θ q2; (θ is non algebraic operator)*

Begin

Partialresult : array of table;

Finalresult :tables; i: integer;

....

I :=1;

eval(q1) ; (Evaluate q1 and store result in Result stack)

for each row r in top(resultstack) do

push( ) (open new scope on Enviornment stack)*

eval(q2) ; (Evaluate q2 and store result in Result stack)

Partialresult[i]= combine (r, top(resultstack)); (create partial result)*

Pop() (pop the current scope)*

Pop(resultstack)

I=i+1;

End;

Finalresult= merage (Partialresult);

End; (case)*

*End;(* eval)*

The SBA semantics is uniform basis for the definition of several non algebraic operators of OQL like language

Selection: q1 where q2 , where q1 is any query and q2 is Boolean valued query . If q2 returned TRUE for the row r returned by q1, then r is the element of final result table; otherwise it is skipped.

Navigation, Projection, path expression: q1.q2 the final result table is the union of tables returned by q2 for each row r returned by q1. This construct covers generalized path expressions, e.g. q1.q2.q3.q4 is understood as ((q1.q2).q3).q4

Dependent join , navigational join : q1σ q2 a partial result for particular r returned by q1 is a table obtained by a concatenation of row r with each row returned by q2 for this r. the final result is the union of partial results.

Quantifiers : q1 (q2) and q1(q2), where q1 is any query , and q2 is Boolean –valued query for the final result is false for at least one row r returned by q1; otherwise the final result true . for actual definition is applied., if q2 returns FLASE

4.5.2 Distributive Property for Identifying sub queries

Here considering non algebraic operators are distributive. The properties is defined below

Definition

A non algebraic operator θ is distributive, if any syntactically correct queries q1, q2, q3 (q1, q2 are union compatible) and for any database holds following

(q1 union q2) θ q3

Is semantically equivalent to

(q1 θ q3) union (q2 θ q3)

In other world, a non algebraic operator θ is distributive , if for any query q1θ q2

Its result can be calculated as union of all result of r θ q2, where r is a row of the table returned by q1.

That is, the final result is always a union of partial results taken for all rows returned by q1.

Proposition I

Operator where, dot, and independent join are distributive

Union of partial result obtained for each row returned by the sub query occurring on the left side of such an operator. Quantifiers are not distributive. There are also others non – algebraic operators that are not distributive (e.g. grouping, ordering) , not consider in this paper.

Proposition II

The following associativity properties hold

1. (q1.q2).q3 is equivalent to q1.(q2.q3)
2. (q1 ⋈ q2) ⋈ q3 is equivalent to q1 ⋈ (q2 ⋈ q3)
3. (q1 ⋈ q2) . q3 is equivalent to q1.(q2.q3)
4. (q1 ⋈ q2) where q3 is equivalent to q1 ⋈ (q2 where q3)

Finding out subqueries

The concept of handing independent subqueries is investigate [] . in our approach we are finding subqueries for handling wider and complex queries.(we called it as complex query , because it combines different queries with non algebraic operator). Handling subqueries with context of sub query result.

Query Q1: Find MatchID of all matches in database in which Tendulkar batted.

This query need to use two classes as the player name is given only in the class Player and batting performance is given only in the lass batting . we will use a subquery to obtain the PID of Tendulkar and use that information to obtain matched of matches that he has played in.

This query is simple use of a subquery. The subquery only return a single constant which is compared with another value in the WHERE clause. If the comparison's result is true then the row in the outside query is selected otherwise it is not. In a query like this in which the subquery returns only value, PID IN could be replaced by PID = but if are not certain that the result of the subquery will be constant, then it is best to use the IN operator.

Batting where PID IN((player where Lname= "Tendulkar").playerID).matchID

In this query, we find subquery

(player where Lname= "Tendulkar").playerID group as P

And transform whole query to following form

((((player where Lname= "Tendulkar").playerID) group as P). (Batting where PID IN .P).matchID)

We decompose query by identifying subqueries , if subquery is matched and proposed as cached query uniquely identified by its reference id ,(reference id , which we created to identify cache result stored in cache to identify query)

Batting where PID IN($cache(reference_id#1).matchID)

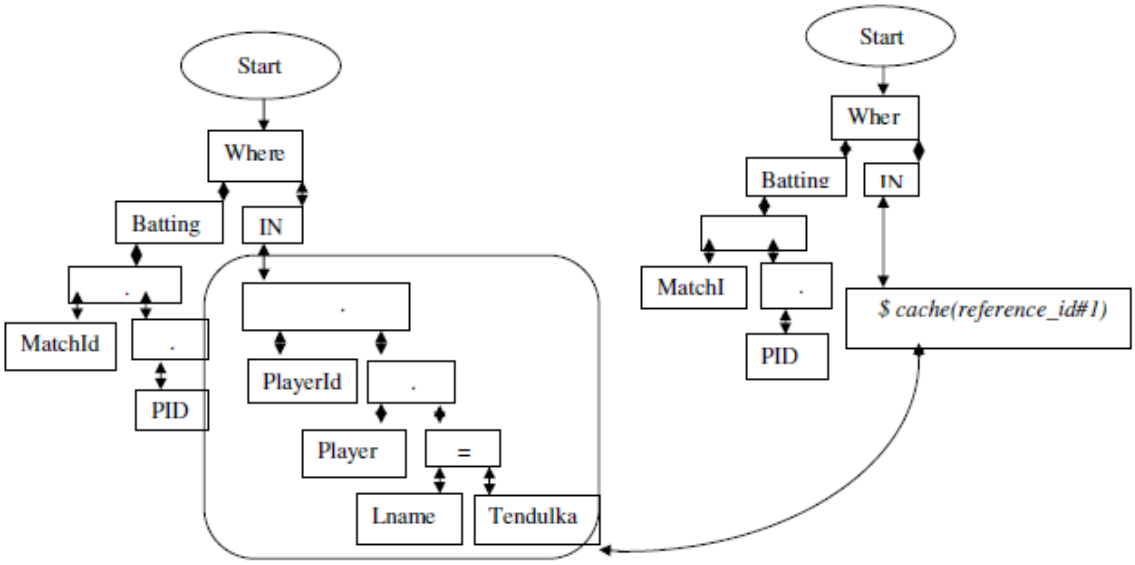


Figure 3: Sample Query Optimization

Using aggregate Function

Query Q2: Find maximum score of Tendulkar and MatchId of all matches in database in which Tendulkar batted.

This query need to use two classes as the player name is given only in the class Player and batting performance is given only in the class batting . it is same query as above but here we need to find Maximum score of Tendulkar , here we write query by using aggregate function *Max* on attribute Nruns .we will use a subquery to obtain the PID of Tendulkar and use that information to obtain matched of matches that he has played in.

This query is simple use of a subquery. The subquery only return a single constant which is compared with another value in the WHERE clause. If the comparison's result is true then the row in the outside query is selected otherwise it is not. In a query like this in which the subquery returns only value, PID IN could be replaced by PID = but if are not certain that the result of the subquery will be constant, then it is best to use the IN operator.

Batting where PID IN((player where Lname= "Tendulkar").playerID).matchID,Max(Nruns)

In this query , we find subquery

Qsub1: (player where Lname= "Tendulkar").playerID group as P

and
Qsub2: max(Bats.Batting.Nruns) group as q

And transform whole query to following form

(((player where Lname= "Tendulkar").playerID) group as P).((Batting where PID IN .P). matched , batted.player.q)

We decompose query by identifying subqueries , if subquery is matched and proposed as cached query uniquely identified by its reference id ,(reference id , which we created to identify cache result stored in cache to identify query)

Batting where PID IN

($cache(reference_id#1).matchID,(batted .player($cache(reference_id#2))

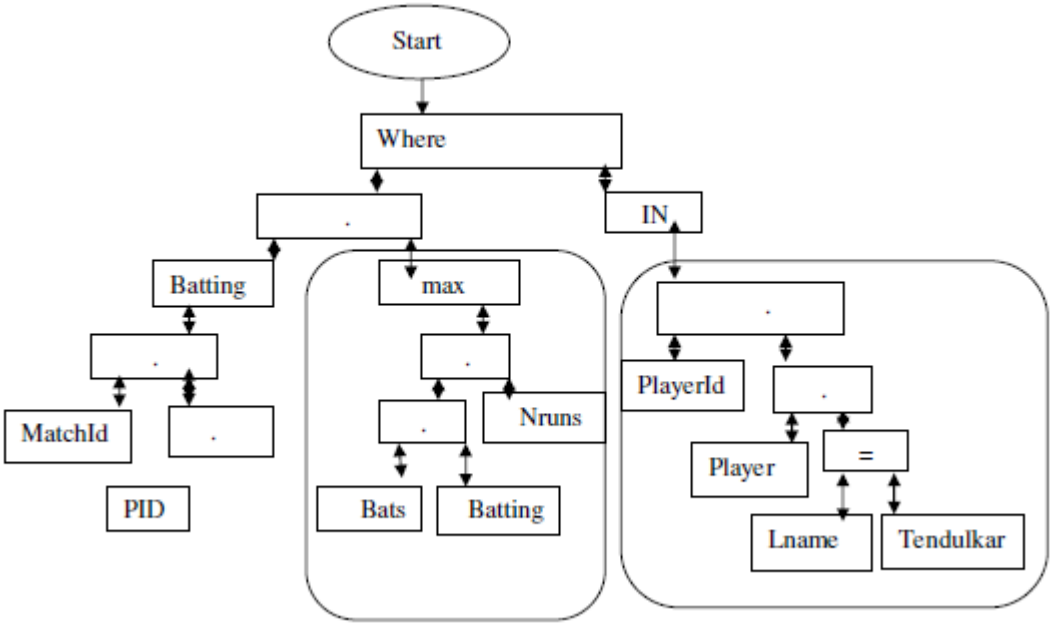


Figure 4 : Sample of Query Optimization , Handling complex queries through Cache Reference.

Subquery is matched and proposed as cached query uniquely identified by its reference id , (reference id , which we created to identify cache result stored in cache to identify query)

($cache(reference_id#1).matchID,(batted .player($cache(reference_id#2))

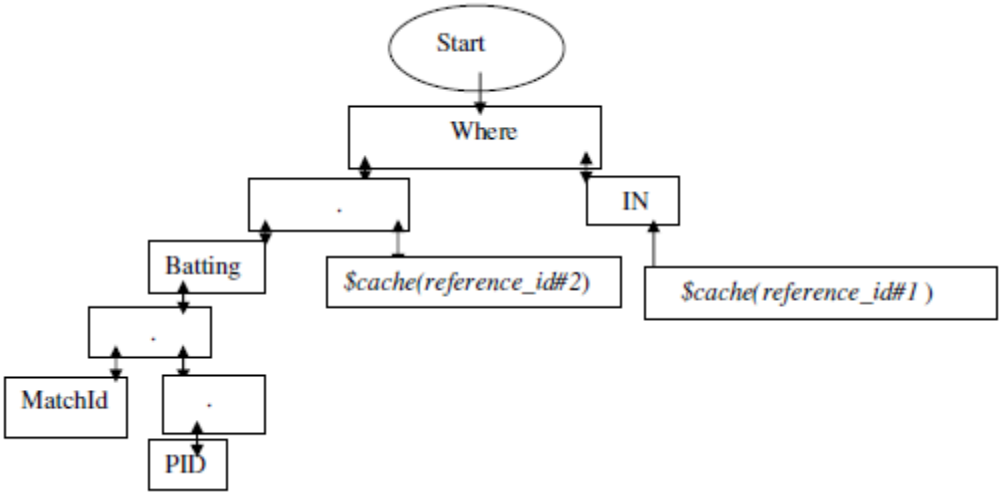


Figure 5: Sample Query Optimization through Reference Cache

5. EXPERIMENTAL RESULT

We have the performance of the optimizer by calculating response times 100 subsequent results using set of queries retrieving data from database containing over 15000 objects. We have compare data with three optimization strategies with cache, without cache, with Db40, and in many cases especially complex queries, wider queries response time were 100 times faster. Table1 shows result in term of computation time (in micro second) of three different approaches of three different complex queries (considering complex queries , which contain many join operation , aggregate functions and many subqueries)

Approach	Query One	Query Two	Query Three
DB4o [no optimization]	150087	144779	117553
Prototype [no caching]	375127	391564	345777
Prototype[with caching]	21000	19845	20221
Note: Numbers indicates time taken to calculate the result of the query in microseconds			

6. CONCLUSION AND FUTURE SCOPE

Based on the experimental results we can state that Decomposition and Caching techniques in Object Oriented Queries have resulted in approx around hundred percent increases in performance and query output.

We have presented a new approach of query optimization, where query execution done using caching of result of previously answered queries; here we try to address future scope of [17]. Issue that we are try to handle is recognizing some parts of cached result helpful for answering other queries and combining many cached queries while producing a result of one wider query.

The work on cached queries is continued. There are many future extensions of this work, some are extending caching solutions for distributed environment.

REFERENCES

[1] Narayanan K.R.S. and Jayanthi T., “Overview of Object Oriented Databases”, Conference on Recent Advances in Information Technology ,READIT - 2005

[2] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham, “Materialized view selection and maintenance using multi-query optimization,” in Proc. of ACM SIGMOD, pp. 307–318, 2001.

[3] Hanna Kozankiewicz, Krzysztof Stencel, Kazimierz Subieta “Distributed Query Optimization in the Stack-Based Approach” Springer Journal Lecture Notes in Computer Science, Volume 3726,pages 904-909, Springer-Verlag Berlin Heidelberg 2005.

[4] Roberto V. Zicari, Editor ODBMS.ORG “A New Renaissance for ODBMSs” ICOODB 09 conference on July 2, 2009, Zurich

[5] Silberchatz ,Korth ,Sudharshan “Database System Concepts” 4th edition Mc-Graw-Hill, ISBN 0-07-120413-X chapter 8, Object Oriented Database pages 307-333, chapter 13& 14 Query processing, Query Optimization,2002.

[6] Antonio Albano, Giorgio Ghelli, and Renzo Orsini "Programming Language of Object Database" in Very Large Data Bases VLDB Journal, Volume 4, 1995, Pages 403-444.

[7] Belal Zaqaibeh and Essam Al Daoud, "The Constraints of Object-Oriented Databases" International Journal of Open Problems in Computer Science and Mathematics, IJOPCM, Volume 1, No. 1, June 2008.

[8] Yannis E. Ioannidis, "Query Optimization" in International Journal on Very Large Data Bases VLDB Journal ,Volume 6, Issue No 2, May 1997 Springer- Verlag New York, USA.

[9] Christian Rich, Marc H. Scholl "Query Optimization in OODBMS" in Proceeding of The German Database Conference BTW., Springer, ISBN 3-540-56487-X March 1993.

[10] S. S. Dhande,G. R. Bamnote "Query Optimization in OODBMS: Decomposition of Query and cached for Wider Query Management" in International Conference on Research and Scientific Innovation ICRSI -2013, by Research and Scientific Innovation Society Australia | New Zealand | India, International Journal of Latest Technology in Engineering Management &Applied Science A Unit of International Standards Publication ISSN 2278 – 2540, Volume III, Issue I, January 2014

[11] Minyar Sassi and Amel Grissa-Touzi "Contribution to the Query Optimization in the Object-Oriented Databases" in Journal of World Academy of Science, Engineering and Technology, WASTE Issue No. 11, 2005.

[12] Bleja, M. Stencel, K. Subieta, K., Fac. "Optimization of Object-Oriented Queries Addressing Large and Small Collections" " in International Multiconference Computer Science and Information Technology, IMCSIT 09. ISBN: 978-1-4244-5314-6, 2009.

[13] Agathoniki Trigoni "Semantic optimization of OQL queries" Technical Reports published by the University of Cambridge ISSN 1476-2986, October 2002.

[14] Piotr Cybula and Kazimierz Subieta "Query Optimization Through Cached Queries for Object-Oriented Query Language SBQL" Springer Journal Lecture Notes in Computer Science, volume 5901/2010, pp.308-320, 2010.

[15] S. S. Dhande,G. R. Bamnote "Query Optimization in Object oriented Database through detecting independent sub queries" in International Journal of Advanced Research in Computer science and software Engineering. (IJARCSS), ISSN: 2277-128X, VOL-2,ISSUE-2 , FEB 2012.

[16] S. S. Dhande,G. R. Bamnote "Query Optimization of OODBMS: Semantic Matching of Cached Queries using Normalization" in International Conference on "Emerging Research in Computing, Information, Communication and Applications"ERCICA-2013, Elsevier Publication DBLP,ISBN:978-93-5107-102-0, Aug 2013.

[17] K. Bratbergsengen, K. Novag " improved and optimized partitioning techniques in database query processing" , springer LNCS 1271, PP 69-83, 1997.

[18] J.M Hellerstein , J. F. Naughton , " Query execution technique for caching expensive methods" , proceeding of SIGMOD , PP 423-434, 1996.

[19] "Next-Generation Object Database Standardization" Date: 27-September-2007 Object Database Technology Working Group White Paper. in International Multiconference Computer Science and Information Technology, IMCSIT 09. ISBN: 978-1-4244-5314-6, 2009.

[20] "On Oracle Database 11g" Oracle Magazine, VOL XXI, No 5 , 2007

[21] Mrs. Laika satish and Dr. Swami Halawani , "A fusion algorithm for joins based on collections in Odra (Object Database for Rapid Application development)." In IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 2, July 2011 ISSN (Online): 1694-0814, PP 289-292

[22] Piotr Cybula, Kazimierz Subieta "Decomposition of SBQL Queries for Optimal Result Caching" Proceedings of the Federated Conference on Computer Science and Information Systems, ISBN 978-83-60810-22-4, 2011 pp. 841–848

[23] K. subieta, C. Beeri, F. Matthes, and J. W. Schmidt, "A Stack Based Approach to query languages", in Proc. of IInd Springer Workshops in Computing, 1995.

[24] K.. Subieta, "Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL)" http://www.sbql.pl/overview/, 2008.

[25] M. Tamer, Jose A. Blakeley "Query Processing in Object Oriented Database System" in Proceeding of ACM SIGSOFT Software Engineering Notes ,volume 35 , Issue No. 6, ISBN:0-201-59098-0, November 2010.

[26] David Dueck, Yiwen Jiang, and Archana Sawhney . "Storage Management for Object-Oriented Database Management Systems: A Comparative Survey"

AUTHORS

S. S. Dhande is currentaly working as Associate. Prof. at Sipna college of Engineering and Technology, Amravati . Maharastra, india. She did her B.E. (comp sci and Engg) and M.E. (Comp Sci and Engg) from S. G. B , Amravati University, she is currently pursuing her PhD in faculty of Engineering & Technology in area of Object Oriented Database Management system. She has published many papers at national as well as international level. She is member of ISTE, IE,IETE, India.

Dr. G. R. Bamnote is Professor and Head of Department of Computer Science and Engineering at PRM Institute of Research and Technology and Dean of Engineering faculty of Amravati university. He did BE from karad, pune university and ME, PhD from Amravati university in 2009. He has published many papers at national as well as international level and guided many students for PG and Phd. He is life member of ISTE, CSI and fellow of IETE and IE of India.