Ms M. C. Nikose, Ms S. S. Dhande, Dr. G. R. Bamnote / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 3, May-Jun 2012, pp.2479-2486

Implementation of Optimization in Object-Oriented Queries

Ms M. C. Nikose[1], Ms S.S.Dhande[2] Dr. G.R.Bamnote[3]
*(Computer Science & Engineering, Sipna's College of Engg & Tech / Amravati University, India)
** (Computer Science & Engineering, Sipna's College of Engg & Tech / Amravati University, India)
*** (Computer Science & Engineering, P. R. Meghe Institute of Tech & Research / Amravati University, India)

ABSTRACT
Query optimization techniques are dependent upon the query model and language. The query model, in turn, is based on the data (or object) model since it defines the access primitives which are used by the query model. These primitives determine the power of the query model. Object-oriented languages allow expressing queries explicitly in the code, which are optimized using the query optimization techniques from the database domain. With respect to this, a formalized object query language (OQL) has been developed that performs optimization of queries at compile time. We use a universal object data model referred to as the stack based approach, which is responsible for naming-scoping-binding principle. In this paper we proposed one of the methods of query optimization depending on rewriting. Query rewriting deal with optimization method at textual level. Optimization by rewriting concerns queries containing so called independent subqueries. It consists in detecting them and then factoring outside the loops implied by query operators.

Keywords - **Query model, OQL, Query optimization, Rewriting, SBA.**

I INTRODUCTION

Nowadays, the necessity to support complex data in databases is intensified. Models trying to answer to these needs appeared as the object-oriented and the object relational model. Relational languages are amplified to a big extent by the idea of declarative query languages, notably SQL. However, the relational model only permits the alphanumeric data management. A similar role in object-oriented database is fulfilled by object query languages. The usefulness of these languages strongly depends on query optimization. The data model of a DBMS lays down the possible structure of the data; to provide easy access to the user, a high-level query language is supported. 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[4].

A query language is special tool used in contemporary database management systems (DBMSs) to retrieve information from data storage. Although finding information is probably the most important application of modern query languages, they are also used to insert, update, and remove data stored in a database, to identify constraints, active rules, etc. According to specialists modern query languages should be, Declarative, of high-level, Macroscopic Independent of the data organization, Interpreted, Efficient.

In modern database query languages usually have those features, these features are very useful. From user point of view, because they make query language user friendly. However, native evaluation of queries defined in such languages may be very inefficient, especially for large database. Therefore, efficient query processing i.e. extraction of information from a database is one of the most desirable features of contemporary DBMSs. From user's point of view a system is a black box, for user the most important criterion is the time needed to evaluate query. Therefore, the main task of query optimization in contemporary DBMSs is to minimize the time, of query evaluation to the level accepted by the user.

A query can be evaluated in many different ways and the difference between the best access plan and the worst one can be very large, sometimes many orders of magnitude. Therefore, efficient query optimization is one of the most important tasks of DBMSs. DBMSs support high performance of query processing by using special data structures (e.g. indexes) or by performing some operations simultaneously. However, the major potential of performance improvement can be achieved by special preprocessing of query before execution. This preprocessing is referred as query optimization, is performed by special module called query optimizer.

Generally approaches to query optimization are subdivided into two main categories with regard to when it is performed. If it is entirely performed before a query is executed, then it is called static optimization. If only a part of it is rendered before query execution and the rest is made during evaluation (i.e. at run time), then it is referred to as dynamic optimization[4].

II QUERY PROCESSING & OPTIMIZATION

Query processing and its optimization have been two of the most popular areas of research in the database community [6]. 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. 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.

An important task in query processing is query optimization**.** Query optimization techniques are dependent upon the query model and language. For example, a functional query language lends itself to functional optimization which is quite different from the algebraic, cost-based optimization techniques employed in relational as well as a number of object-oriented systems. The query model, in turn, is based on the data (or object) model since the latter defines the access primitives which are used by the query model. These primitives determine the power of the query model. Usually, user languages are high-level, declarative languages allowing to state 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. Usually, one has to accept just feasible execution plans, because the number of semantically equivalent plans is too large to allow for enumerative search.

2.1 QUERY PROCESSING ARCHITECTURE

During static analysis we simulate run-time query evaluation to gather information that we need to optimized the queries. The general architecture of query processing is shown below. A parser of queries and programs takes a query source as input, makes syntactic analysis and returns a query/program syntactic tree. A query/program syntactic tree is a data structure which keeps the abstract query syntax in a well-structured form, allowing for easy manipulation (e.g. inserting new nodes or subtrees, moving some subtree to another part of the tree, removing some subtrees, etc.). Each node of the tree contains a free space for writing various query optimization information.

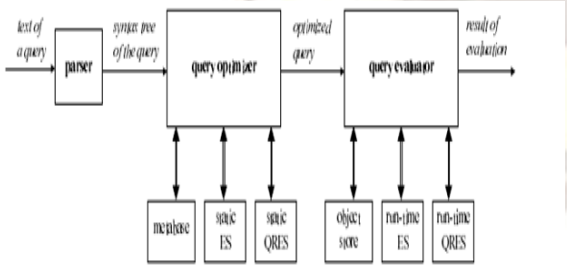


Fig. 1. Architecture of query processing

Syntactic trees are the most convenient way to represent and to process query that can be optimized through rewriting to be used during query evaluation. Static analysis involves,

- Metabase (counterpart of an object store), which is a formal data structure representing database schema.
- Static environmental stack (counterpart of a run-time environmental stack) that stores so-called static binders, i.e. named signatures. The structure during compile time simulates all the operations that are made on the environmental stack by query/program run-time mechanism.
- Static result stack: the structure that accumulates signatures being the type description of corresponding run-time temporary and final query results.

Query processors are usually combination of two components namely,

- Query parser: checks whether a query is syntactically and semantically correct, and transform it to some internal form.
- Query optimizer: determines query evaluation plans. A query optimizer should generate a set of plans, among these only one of the plan is chosen.
- Query plan evaluator: evaluates query in accordance with the chosen evaluation plan.

III STACK-BASED APPROACH (SBA)

SBA is a formal frame addressing object-oriented database query and programming languages (PLs). The approach is motivated by the belief that there is no definite border line between querying and programming; thus there should be a universal theory that uniformly covers both aspects. SBA[5] offers a unified and universal conceptual and semantic basis for queries and programs involving queries, including programming abstractions such as procedures, functions, classes, types, methods, views, etc. SBA [7] treats query language as a special kind of programming language; it is an attempt to build a uniform semantic foundation for integrated query and languages. SBA allows to precisely determining the semantics of query languages; there relation with object oriented concepts, with imperative programming constructs and with programming abstraction, including procedures, functional procedures views, modules etc. its main features are the following

- The naming-scoping-binding principle is assumed, which means that each name occurring in a query is bound to the appropriate run-time entity (an object, attribute, method, parameter, etc. according to the scope of this name).
- One of its basic mechanisms is an environmental stack (ES). The stack is responsible for scope control and for binding names. In contrast to classical stack it does not store objects, but some structures built upon object identifiers, names, and values.
- In contrast to relational languages and OQL, the relativity principle is assumed, i.e. the

syntax, semantics, and pragmatics are identical at an arbitrary level of data hierarchy.

- Types are a mechanism to determine whether objects are built in a proper way (i.e. in accordance with the database schema).
- For objects the principle of internal identification is assumed (i.e., each run time entity has a unique internal identifier).

IV ABSTRACT OBJECT-ORIENTED STORE MODEL

An object store model is simply an abstract view on data structured stored in the database and is orthogonal to any ideologies such as the relational model or XML. The main components of object store model are its location, the name that can be used to denote it, the value stored there i.e. its contents. Atomic values, records ,conceptual object are the example of object store, which resides somewhere in memory and needs location for internal identiction to perform operation like update, retrive, delete. These location does not have meaning or structure,instede they can be used to access object. Location are some times called as identifires[5].

In SBA each object has the following features;

- **Internal object identifier (OID):** It is used to identify internally an object from the object store model; in particular, it will be used as a reference or pointer to an object. For each object stored in the object store its internal object identifier is unique.
- **External object name**: They are explicitly assigned to objects by a database designer, a database administrator, an application programmer, or another human agent, e.g. *Person*, *Department*, etc.
- **Contents** of an object, which can be a value, a link or a set of objects.

Following three sets are used to define object,

- set of internal identifiers
- N- set of external names
- V- set of atomic values

Formally let, i, i1, i2 Є I, n Є N and v Є V. Objects are formulated as three triples as,

- Atomic object: <i, n, v> where i is the ID of the object, n is an external name assigned to the object, and v is the value of the atomic object (e.g. an integer, a string, etc.)
- Link object: <i1, n, i2 > where i 1 is the ID of the reference object, n is an external name assigned to the object, and i2 is the ID of the object referred to.
- Complex object <i, n, S> where i is the ID of the object, n is an external name assigned to the object, and S is a set of objects.

V ENVIRONMENTAL STACK

Environmental stack (ENVS) is a basic concept of semantics and implementation of majority of well-known programming languages. The stack employs following roles

- Control over scopes of variable names occurring in a program and binding these names to run-time entities;
- Storing values of local variables of procedures;
- Storing values of actual parameters of procedures;
- Storing a *return track*, i.e. an address of the program code where the control should be returned after the given procedure is terminated (usually the next address after the procedure call).

ENVS is subdivided into sections, which are ordered, with newest section known as top and oldest one as bottom. A section is associated with a particular *procedure call* or an executed *program block*. When the control is transfer to a procedure call, a new section of volatile objects (activation record) is pushed on the top of the stack. The section is popped from the sack when the procedure or program block is terminated. For the procedure that is currently running all values of parameters, local variables, objects and any other local entities are stored within the top stack section. Activation record possess the abstraction principle, which allows the programmer to consider the currently being written piece of code to be independent of the context of its possible uses. The stack allows to associate parameters and local variables to particular procedures invocation. The stack also used to accomplish strong typing, encapsulation, inheritance and overriding. In SBA the stack has a new function: processing queries acting on the object store. It makes it possible to control scopes of all names occurring in a query in a simple and uniform way. It also makes it easier to understand the precise semantics of queries[1], [5].

5.1 Concept of Binding

The mechanism, which make it possible to determine the meaning of each name is called binding. The sections of ENVS consist of entities called binders whose role is to relate a name with a run time entity (object, procedure, view). Binding of the name *n* means that the query interpreter looks through the consecutive sections of the ENVS to find the closest binder with the name *n*. Binding is performed on ENVS according to "search from the top" rule. A binder is a pair *(n, x)*, where *n* is an external name, and *x* is some value (reference to object). The pair is written as *n(x)*. A binding action for some name *Y* is performed according to the following steps:

- The machine checks the top of the stack for an entity named *Y*; if there is such an entity, the binding returns it as the result and the action is terminated.

- If the top does not contain an entity named *Y*, a section below the top is checked.
- Such a process is continued in lower and lower stack sections, till the entity named *Y* is found. Visiting particular stack sections is governed by *scoping rules* that require omitting some sections;
- If *Y* is not found on the stack, then the global environment is searched. The global environment contains static variables, database objects, computer environment variables, procedure libraries, etc. Alternatively, we can assume that the global environment is the lowest stack section in such a case *Y* must be found on the stack, otherwise an error should be reported.

5.2 Opening a new section of ENVS

In SBA at the beginning of a user session ENVS contains of single section containing binders for all root database objects. During query evaluation the stack is growing and shrinking according to query nesting. The final ENVS state is exactly the same as the initial state. Opening a new scope on the environment stack is caused by entering a new procedure (function, method) or entering a new block. Respectively, removing the scope is performed when the control leaves the body of the procedure or the body of the block. To these classical situations of opening a new scope on the environment stack we add a new one. It is the essence and motive of SBA.

The idea is that some query operators which combines queries behave on the stack similarly to program blocks. These operators are divided into algebraic and non-algebraic. The main difference between them is whether they modify ES during evaluation or not. An operator is algebraic if it does not modify the state of ENVS. The algebraic operators include numerical and string operators and comparisons, Boolean and, or, not, aggregate function, and sequence operators and comparisons, structure constructor, dereferencing etc. Operators which name a query result are unary algebraic operators too. If query q1 Θ q2 involves a non-algebraic Θ, then q2 is evaluated in the context of q1. The context is determined by the new section opened by the Θ operator on ENVS for an element of q1. A new stack section pushed onto ENVS is constructed by a special function. Subqueries q1 and q2 cannot be processed independently, the order of evaluation is important. Non-algebraic operators include projection /navigation (q1.q2), selection (q1 where q2), dependent join (q1 join q2), quantifiers (∃q1q2), transitive closure and ordering.

The following figure present all the stages of the evaluation of simple query P.getSal()>2500, here the part getSal>2500 behaves like a program block executed in the environment consisting of the interior of an professor object.

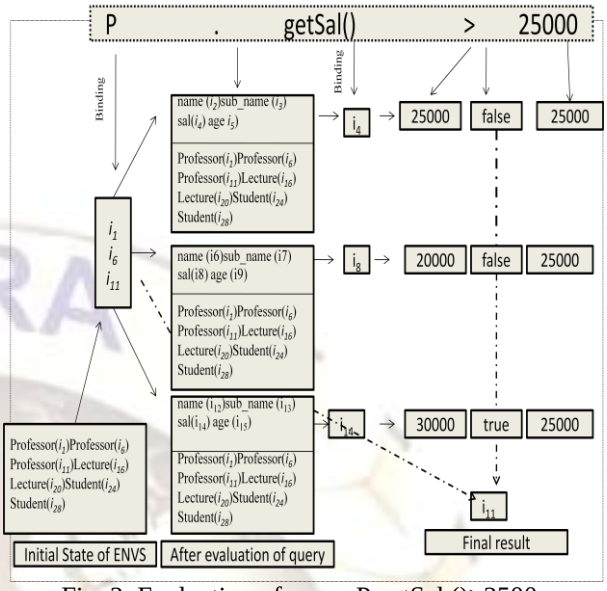


Fig. 2: Evaluation of query P.getSal ()>2500

VI STATIC ANALYSIS OF QUERIES

Static analysis is a compile-time framework of static optimization. It performs the following tasks,

- Type checking: each name and each operator is checked according to the class hierarchy.
- Each name occurring in query is assigned to ENVS section, which is relevant for binding name.
- Generating syntax trees of queries which are then modified by query rewriting methods.

6.1 Static Analysis Algorithm

In our project we are assume that queries can be invoked for an arbitrary state of the object store and ENVS, thus at compile time we cannot predict precisely. So the algorithm of run-time evaluation is modify little to analyze queries statically. Generally, the algorithm of static analysis works in the same way as algorithm of run-time evaluation with some changes as,

- Operations are performed on signatures rather than of run-time entities
- The signature of the query l, where l is literal, is inferred type of l, e.g. integer, Boolean, string etc.
- The result signature of the query is n, where n is name, is inferred from static ENVS as the system binds name n (by looking static binder n(x)) on static ENVS. Resulting signature is x.
- If the query is of the form q as n, and q has the signature <q_sig>, then the query has the signature n(q_sig).
- If a query has the form q1 Δ q2, where Δ is algebraic operator, then signature of query is a composition of signatures of its components,

according to the type inference rule for Δ operator. E.g. if q1 has the signature integer, q2 has real and the operator is + then resulting signature is real.

- The signature of query q1 Θ q2, where Θ is non-algebraic operator, is composed of the signature of q1 and q2 in the way determined by the Θ operator.
- The signature of a new section pushed onto the static ENVS stack is built based on the elements of the signature of the static value processed by a non-algebraic operator in away similar to that at run time. For instance, if the signature of the value is ref(student), then the signature of the new ENVS section is{name(ref(name)), subject(ref(subject)), grade (ref(grade))}. Similarly for signature storing class properties.

VII METHOD OF INDEPENDENT SUBQUERIES

In this paper we are proposing one of the methods of query optimization. The idea of this method is based upon the observation that, if none of the name in subquery is bound in the ENVS section opened by the non-algebraic operator currently being evaluated, then this subquery is independent of this operator. Subqueries are called independent if they can be evaluated outside loops implied by the non-algebraic query operators. Such subqueries are worth analyzing because they usually imply optimization possibilities [1],[2].We use a special technique called the method of independent subqueries to optimize queries. This method is based on query rewriting that deals with optimization method at textual level. Rewriting means transforming a query q1 into a semantically equivalent query q2 promising much better performance.

Technically, it consists in analyzing in which sections particular names occurring in a query are bound. It turns out that if none of the names in given subquery is bound in the scope opened by the non-algebraic operator currently being evaluated, then that subquery can be evaluated earlier than it results from its textual place in the query it is a part of. The method modifies the textual form of a query so that all its subqueries will be evaluated as soon as possible. To determine in which scopes names occurring in a query are bound at run time [9], we statically analyze that query and during analyzing we additionally do the following:

- Each non-algebraic operator is assigned the number of the scope it opens,
- Each name in the query is assigned two numbers:
  - The stack size: the number of scopes thst is on static ENVS when the binding of this name is being performed.
  - The binding level: the number of the scope on the stack in which this name is bound.

All those numbers are determined relatively to the bottom scopes of a query.

VIII EXPEREMENTAL SETUP AND RESULT

Query optimization is the process of finding the best or rather a reasonably efficient execution plan, thus by minimizing the time of query evaluation & the cost of evaluation to the level accepted by user. The proposed method optimizes the object oriented queries in order to reduce the time required to execute a query. Here we consider an object-oriented database db4o. It provides all the benefits of an OO environment including data abstraction, inheritance, and encapsulation. The object model simplifies maintenance and refactoring and seamlessly integrates with the modern programming languages (Java, .NET)[10].

The db4o object database engine consists of one single jar file. To install db4o we have to add one of the db4o-*.jar files to your CLASSPATH. Here we are using an integrated development environment like Eclipse SDK, we need to copy the db4o-*.jar to the lib folder of our project and then add db4o to our project as a library. The experiment is performed on the Intel core i3 processor 4GHz system running windows7 Home Basic with 4 GB RAM.

The entire process of query optimization is proposed as follows:

1. First take a query as an input
2. Comparison OODBMS (db4o) & RDBMS (MySQL) in terms of time.
3. Rewriting step

Example Database

The fig3 shows the class hierarchy of an example database it defines three classes as Student, Professor, and Lecture. The name of the class is followed by cardinality. All objects and class properties are public. Student, Professor, and Lecture are the root objects. Fig4 shows tiny database built upon the schema presented in fig. To store class properties we use special objects. Links are denoted by arrows, and inherited links are denoted by thick arrows.

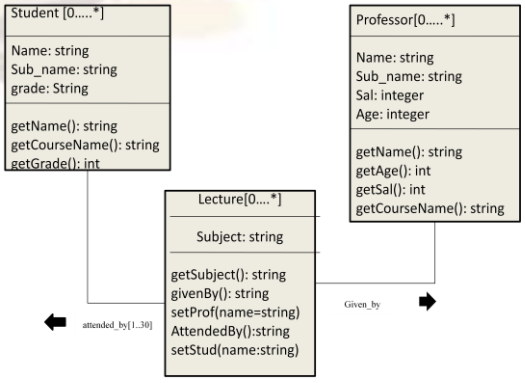


Fig.3: Class Hierarchy

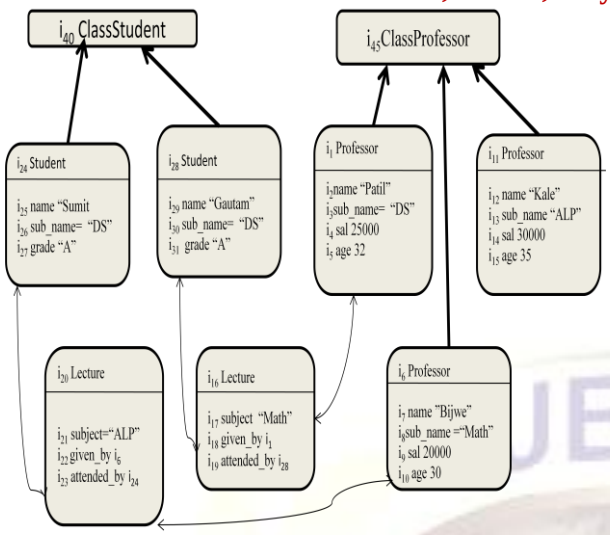


Fig. 4: Tiny Database

Step I: Taking query as input

Consider a query, which retrieves and count number of student whose course name is same as that of the lecture taught by the professor.

Return

lect..attendedBy().getCourseName().equals("Physics")
(1,1)2 (2,2) 3 (3,1) (1)
&& lect . givenBy( ) . getName. equals("Rathi")
(1,1) 4 (4,2) 5 (5,5)
&& lect . getSubject1(). equals("physics")
(1,1) 6 (6,1)

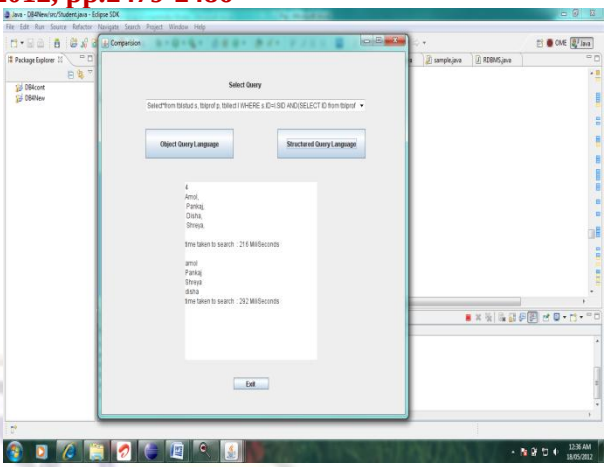
Notice that when an evaluation of its subquery (getting name of professor giving a lecture)

lect . givenBy(). getName(). name (2)

At starting there are two non-bottom section on ENVS: one is open by internal dot operator and other is by external dot operator. Subquery (2) is independent of non-algebraic operator (dot), since all names in this subquery are bound either in section 4 or section 2, while that operator opens section 3. However it is dependent on internal dot operator, which opens section 2, since givenBy is bound at section 2. Subquery (2) can be evaluated separately as it is independent of external dot operator.

Step II: Comparison of Query generation parameter in terms of Query Languages (OQL with SQL)

The query taken in first step is executed in both db4o and MySQL database. The following snapshot shows the time required to execute a query in both the database. From this snapshot it is clear that db4o takes less time as compare MySQL.



Screenshot 1: comparison of query languages

Fig.6 shows performance of proposed query evaluation of db4o with MySQL in terms of time.

Fig 5 . Run time of queries

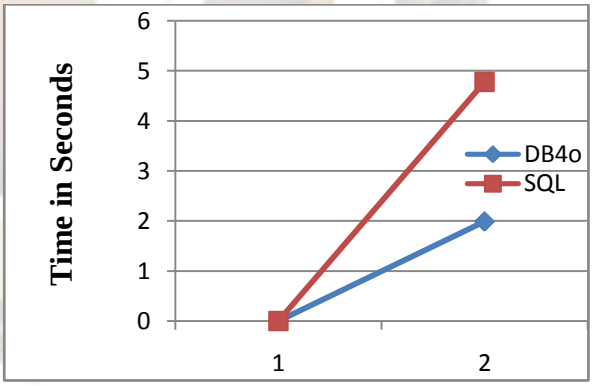


Fig.6 comparison of queries

Step III: Rewrite step

Rewriting means transforming a query q1 into a semantically equivalent query q2 promising much better performance. It consists in detecting parts of a query matching pattern. When it is recognized, a query is rewritten according to the predefine rewriting rule. Such optimization is compile-time action entirely performed before query is executed, hence query

optimization process itself does not burden the performance. Rewriting requires performing a special phase called static analysis. During the static analysis we simulate run time query evaluation to gather all the information that we need to optimize queries

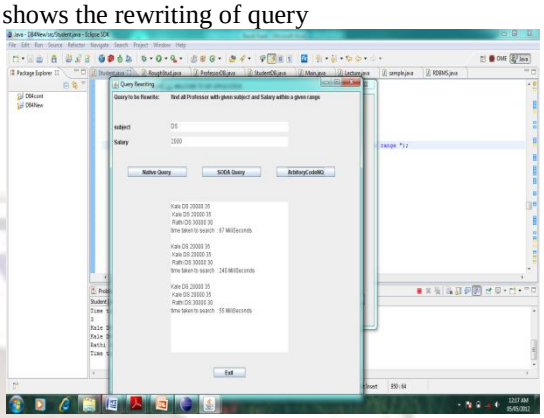
DB4o supports three query languages[10] as

1. **Query by Example:** This is special query method. These are appropriate as a quick start for users who are still acclimating to storing and retrieving objects with db4o, but they are quite restrictive in functionality such as Advanced query expressions,(AND, OR, NOT, etc.) cannot performed, Cannot constrain on values like 0 (integers), "" (empty strings), or nulls (reference types). Basically you pass in a example object to db4o. Then db4o searches the database for all objects which look alike.
2. **Native Queries:** Native queries are the main db4o query interface and they are the recommended way to query databases from your application. Because native queries simply use the semantics of your programming language, they are perfectly standardized and a safe choice for the future. It provide the ability to run one or more lines of code against all instances of a class. Native query expressions should return true to mark specific instances as part of the result set. db4o will attempt to optimize native query expressions and run them against indexes and without instantiating actual objects, where this is possible. In principle you can run arbitrary code as native queries

db4o tries to analyze native queries to convert them to SODA. This is not possible for all queries. For some queries it is very difficult to analyze the flowgraph. In this case db4o will have to instantiate some of the persistent objects to actually run the native query code. db4o will try to analyze parts of native query expressions to keep object instantiation to the minimum.

3. **SODA Queries:** The bytecode is analyzed to create an AST-like expression tree. Then the flow graph of the expression tree is analyzed and converted to a SODA query graph. First the signature of the given class is analyzed to find out the types. This is used to constrain the type in the SODA-query. When the operations a simple and easy to convert, it will be transformed to complete SODA query SODA acts like a filter on all stored objects. But usually you're only interested for instances of a certain type. Therefore you need to constrain the type of the result. You can add constrains on fields. This is done by descending into a field and constrains the value of that field. By default the constrain is an equality comparison.

A query in first step can be rewrite into three languages provided by db4o, in order to improve the performance of query optimization. Since the three languages provided by db4o uses different semantics, the query execution result is same but varies in performance. Fig shows the rewriting of query



Screenshot 2: Executing Rewritten query

Table 1: run time of qurey

Query Language	Time in Seconds
SODA Query	0.36
Native Query	1.76
Arbitory code NQ	0.51

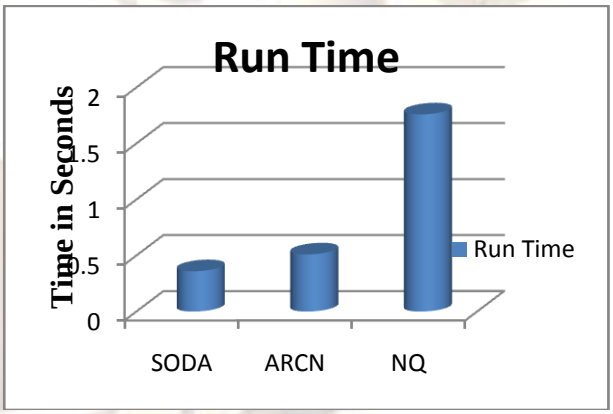


Fig.8 Run time of queries

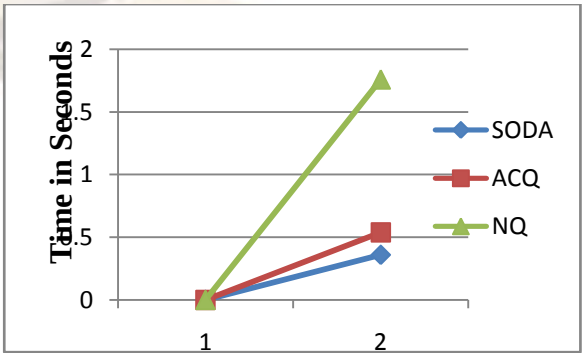


Fig. 9 comparison of queries

IX CONCLUSION

One of the biggest problems in Object Oriented Database is the optimization of queries. Due to these problems optimization of object-oriented queries is extremely hard to solve and is still in the research stage. Finding the optimal strategy is usually too time-consuming except for the simplest of queries and may require information on how the files are implemented and even on the contents of the files, information that may not be fully available in the DBMS catalog. Hence, planning of an execution strategy may be a more accurate description than query optimization. This proposed work is expected to be a significant contribution to the Database Management area which will not only reduce time or efforts but will also improve the quality.

REFERENCE

[1]. **[JPlod00]** J. Płodzień, A. Kraken, "Object Query Optimization through Detecting Independent Subqueries", *Information Systems*, Elsevier Science, 25(8), 2000, pp. 467-490.

[2]. **[Mich09]** Michel Bleja, Krzysztof Stencel, Kazimierz Subeita, Optimization of Object-Oriented Queries Addressing Large and Small Collections, Proc. Of the IMCSIT, 2009, ISBN 978-83-60810-22-4, Vol. 4, pp. 643-680.

[3]. **[Venk10]** Venkata Krishna Suhas Nerella, Swetha Surapaneni, Sanjay Kumar Mardria, Thomos Weigert, Department of Computer Science, Missouri Univarsity of Science & Tech, Rolla, MO, 34 th Annual IEEE Computer Software & Applications Conference,2010.

[4]. [Plod00] J. Plodzien, "Optimization Methods in Object query Languages", Ph. D. Thesis, Institute of Computer Science, Polish Academy of Sciences, 2000.

[5]. **[Subi95]** K.Subieta, C.Beeri, F.Matthes, J.W.Schmidt. *A Stack-Based Approach to Query Languages*. Proc.2nd East-West Database Workshop, 1994, Springer Workshops in Computing, 1995, 159-180.

[6]. **[Adam08]** R.Adamus, M.Daczkowski, P.Habela, K.Kaczmarski, T.Kowalski, M.Lentner, T.Pieciukiewicz, K.Stencel, K.Subieta, M.Trzaska, T.Wardziak, J.Wiślicki: Overview of the Project ODRA. Proceedings of the First International Conference on Object Databases, ICOODB 2008, Berlin 13-14 March 2008, ISBN 078-7399-412-9, pp.179-197.

[7]. **[Subi10]** K.Subieta. Stack-Based Architecture (SBA) and Stack-Based Query Language(SBQL). http://www.sbql.pl/, 2010.

[8]. **[CIDe92]**Cluet, C. Delobel. A General Framework for the Optimization of Queries. Proc. Of SIGMOD Conference, 383-392,1992.

[9]. **[MAG05]** Minyar Sassi, and Amel Grissa-Touzi Contribution to the Query Optimization in the Object-Oriented Databases" Volume 6 ISSN, June 2005 1307-6884

[10]. **[db4oOSODB]** db4objects 11.2008, db4o Open Source Object Database.