Query Optimization in Object-Oriented Database Systems: The *REVELATION* Project

Goetz Graefe
David Maier

Oregon Graduate Center
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999 USA

Technical Report No. CS/E 88-025

July, 1988

Report published in *Advances in Object-Oriented Database Systems: Proceedings of the 2nd International Workshop on Object-Oriented Database Systems,* September 1988.

Abstract

We are exploring a scheme that allows optimizing queries over object-oriented databases with encapsulated behavior. Objects and classes will be able to reveal their behavior in terms of expressions in an algebraic language interpreted by a structural object-oriented database system. An object or class can agree or refuse to reveal its behavior. The structural algebra is richer than relational algebra as it includes operators on complex object collections, and updates and traversals of individual objects. Objects may reveal to the optimizer the structural access paths used by their procedures or cost and other statistics useful for query optimization. The main features of our approach is that the object-oriented user interface language is able to perform general computation and to preserve the encapsulation envelope around classes and types.

Query Optimization in Object-Oriented Database Systems: The *REVELATION* Project

Goetz Graefe
David Maier
Oregon Graduate Center
graefe@cse.ogc.edu, maier@cse.ogc.edu

Abstract

We are exploring a scheme that allows optimizing queries over object-oriented databases with encapsulated behavior. Objects and classes will be able to reveal their behavior in terms of expressions in an algebraic language interpreted by a structural object-oriented database system. An object or class can agree or refuse to reveal its behavior. The structural algebra is richer than relational algebra as it includes operators on complex object collections, and updates and traversals of individual objects. Objects may reveal to the optimizer the structural access paths used by their procedures or cost and other statistics useful for query optimization. The main features of our approach is that the object-oriented user interface language is able to perform general computation and to preserve the encapsulation envelope around classes and types.

1. Introduction

For engineering applications like CASE and CAD systems, none of the currently used file systems, conventional database systems, or custom data managers are totally adequate to the task, because each of them lacks in either modelling power, query optimization, performance on large datasets, or data management services such as authorization, concurrency control, and recovery. File systems put all the burden of mapping complex objects to storage formats on the application programmer, and provide no or only rudimentary support for data abstraction and data management services. Conventional database systems provide concurrency control, recovery, and efficient set-oriented processing, but only for a limited data model and with inadequate performance on complex objects. Custom data managers can give the appropriate data types for an application but usually cannot be modified by the application developer to add new data types as an application evolves, and are often deficient in data management services and in query optimization capabilities. What is needed is a database system with richer data

modelling power and the required high performance on both set and object operations.

The goal of the *REVELATION Project*[1] at the Oregon Graduate Center is to combine the advantages of object-oriented and relational technology in an extensible database system. There is, however, a fundamental difference between object-oriented systems on one hand and relational and extensible systems currently under development on the other hand. In object-oriented systems, algorithms and processing methods are associated with the individual objects and classes. A query is evaluated by sending a message to an object or group of objects. The actual processing steps performed, i.e., the implementation of the method invoked for the message, are beyond the message sender's control. Encapsulating implementation and control of processing in the class or type is a very powerful abstraction mechanism, and contributes considerably to the success and acceptance of the object-oriented paradigm. In relational and extensible systems, query execution is controlled by a central module, often called *scheduler* or *access plan interpreter*. This module employs a limited, fixed set of algorithms, e.g., algorithms implementing relational algebra operators, in a query specific combination.

In the remaining sections, we lay out our plan to combine previous approaches in a new database management system. In Section 2, we provide an brief overview of related work. Section 3 outlines our approach to query optimization in object-oriented database systems. Sections 4 and 5 describe a preliminary database architecture of a high-performance object-oriented database system, in particular the query optimization component and the query evaluation component. Section 6 contains a summary and our conclusions.

---

[1] The reason for the name will be apparent later. Name relationships to the EXODUS project at the University of Wisconsin — Madison and the GENESIS project at the University of Texas at Austin are "purely incidental."

2. Previous Work

Object-oriented database systems are a new and conceptually powerful alternative to existing systems. They support structures and features that are not provided in conventional data models. Among these features are complex objects, object identity, encapsulation of methods and behavior, hierarchies and inheritance of data types, versions of objects, and very large objects [Copeland1984a, Manola1986a, Maier1986a, Maier1986b, Skarra1986a]. These concepts make object-oriented programming and database systems well suited for engineering applications, including CAD, CAM, and CASE. Unfortunately, object-oriented systems do not provide the search performance on large datasets required by these application areas. Engineering data need to be accessible efficiently both in flexible "edit" mode and in large-volume set-oriented "report" mode, as for the relational data model.

Relational systems, on the other hand, place the control solely with an execution module or run-time system, which exists independently of the data. The data in a relational database are purely passive input to algorithms implementing the relational algebra. This view of database management systems, i.e., a general run-time system acting on passive data, is the precondition of query optimization as it is found in existing database systems.

In the last few years, a new direction in database systems research has concentrated on extensibility [Batory1986a, Carey1985a, Carey1986a, Rowe1987a, Schwarz1986a, Stonebraker1986a]. While some of these systems provide extensions and extensibility to basically relational systems, other systems are experimenting with a "toolkit" approach, i.e., software tools and libraries to generate a database system specific to the application domain.

Query optimization has become a wide field of research. Most of the work has focused on relational systems, probably due to the available theoretical framework and to the widespread use of database management systems based on the relational model. A new research direction in database query optimization uses rule-based transformations to determine optimal access plans from algebra expressions. The EXODUS optimizer generator [Graefe1987a] creates a

query optimizer from a rule-based description of the algebra to be optimized. The algebra, i.e., the set of operations and the equivalences of algebraic expressions, is independent of the optimizer generator and can be extended by augmenting an algebra description file. Optimization is performed using an expert system search using the equivalence relationships between algebra expressions and cost functions associated with implementation methods for algebra operators.

Other research groups are also using rule-based approaches to query optimization, e.g., Freytag and Lohman are implementing a rule-based query optimizer for the Starburst extensible (relational) database system [Freytag1987a]. Currently, they are defining a rule language and implementing an interpreter for this language with a built-in search engine [Lohman1988a].

3. Revealing Behavior

Our plan for combining the two approaches is as follows. Behavioral abstraction and encapsulation on the logical and application interface level are strictly retained, but the execution model should employ set-at-a-time algorithms and query optimization as much as possible. When a query entered on the interface level is mapped to an appropriate form for the execution level of the database system, the interface determines which message must be sent to which object or class in order to answer the query. Instead of sending this message, however, the objects and classes involved are requested to **reveal** execution information prior to execution. If an object or class refuses to respond to a *reveal* message, the query is evaluated by sending the original message to the object or class. In the sequel, the original message will be called the *evaluate* message. If, however, an object or class does reveal information, it provides an equivalent processing expression and the means to obtain appropriate input for the expression. The processing expression is formed in terms of an algebra designed for a structural object-oriented database system. It is a superset of the relational algebra, with suitable extensions for large and complex objects. The operators of this algebra can be augmented with procedures for arithmetic, predicates, string processing, and formatted I/O. Typically, the revealed expression will not be composed entirely of algebraic operators. Rather, the expression will involve a small

number of object-message pairs whose behavior is still encapsulated. Further reveal messages can be sent to these objects in an attempt to transform more of the query into algebraic form. The inputs for the expression are objects or classes and evaluate messages with their parameters to be sent to the objects or classes to obtain the input. These are typically the same messages that the revealing object or class would have used internally to obtain this input.

The revealed expression includes sufficient information about the object's or class's behavior to evaluate the original query. Thus, behavior encapsulated in the class definition is now duplicated, and evaluation of the revealed expression can treat the object as passive, i.e., with no behavior attached to it. Obviously, once information is duplicated, complex problems arise in case of change. For the remainder of this prospectus, we assume that the type definition and behavior do not change until the query has been evaluated, or a suitable notification mechanism is in place. Hence, structural interpretation of a algebraic expression can be substituted for the encapsulated behavior.

If the revelation step is used repeatedly, a number of operations can be revealed, resulting is a complex algebra expression or operator tree. Such trees will be evaluated by a rather conventional query evaluation module. The leaves of this tree represent evaluate messages. Since we expect that the storage manager on which the object system is built will be the same as the one used by the query evaluation module, it is possible that a class reveals that an evaluate message is equivalent to scanning a file or an index. Thus, by using the *reveal* message repeatedly, possibly an entire query can be transformed into an algebra expression.

An algebra expression provides three advantages over the original evaluate message. First, by using set-at-a-time algorithms, we can expect much better performance. If a query is expressed in an algebra expression acting on "passive" data, alternative processing algorithms, possibly distributed and parallel versions, concurrency control, and recovery can be introduced transparently to the query and the database objects. Second, complex object traversal can be pre-planned in order to minimize disk accesses and effort for mapping between object represen-

tations on the physical and logical level. Third, this expression can be submitted to a query optimizer.

4. Query Optimization

The optimization paradigm that we have developed in earlier work [Graefe1987a] will be used to optimize revealed algebraic expressions. A program created by the EXODUS optimizer generator optimizes and transforms algebraic expressions (operator trees) and selects implementation methods based on anticipated execution cost. This optimization paradigm is very suitable for the problem at hand. The initial query starts as a single message to a object or class. Looking at it from a different viewpoint, the algebra expression before optimization consists of a single evaluate message. Through several revelation steps a tree is built by expanding leaves. These steps can easily be modelled as tree transformations. When all leaves have been replaced by scan methods or all leaf objects or classes refuse to reveal processing information, the actual optimization phase begins[2]. The operations are reordered according to equivalence rules and heuristics, and algorithms are chosen for the operations. The optimization process and the search strategy employed by optimizers built with the EXODUS optimizer generator is described in more detail in [Graefe1987a, Graefe1987b, Graefe1987c].

5. Query Evaluation

After a query has been optimized, it can be executed by the run-time system. The run-time system will consist of three components, the object processor, the set processor, and the file manager.

The object processor interprets messages to objects and their implementation, for both permanent, disk-based objects and temporary objects. This component is crucial in providing the semantical and computational power expected from an object-oriented system. Since the

[2] The EXODUS optimizer generator allows segmentation of the rule set and to perform query optimization in several phases.

objects at this level can exhibit behavior by procedures attached to the class or an object itself, the object processor provides the power of a general-purpose programming language, thus allowing for user interaction, screen display, etc.

The set processor interprets algebraic expressions as revealed to and optimized by the optimizer. It consists of an extensible set of processing modules. These modules provide and use a uniform item-at-a-time interface. Items can be either single records, objects, or groups of those. By provinding *send* and *receive* operations using this interface, to be inserted into an algebra expression by the optimizer, the location of storage and processing can be made transparent to all other operations, thus allowing for easy extensibility even in distributed and workstation/server architectures. An object or class that can appear in a leaf of an query evaluation tree provides the same interface using the object processor, allowing for a smooth and efficient interface between object processing and set processing.

The algebra appropriate for query optimization and execution is the initial focus of our research. It has to satisfy several criteria. First, it must be powerful enough to express queries originally expressed in an object-oriented programming or query language. At the least, it must encompass the relational algebra and allow for traversal of complex object structures. Second, it must be simple enough to allow efficient search for alternative expressions by the query optimizer. Third, the operations must allow using efficient algorithms to process large sets of objects. Fourth, the algebra must have a solid theoretical foundation such that extensions can safely be made.

Currently, we plan on using relation algebra operators suitably generalized for complex objects. Furthermore, we will design and use an operator to compose complex objects from object id's or object fragments, and operators for object manipulation and transformation.

The same interface is provided by the file manager, which consists of fairly conventional software to efficiently store and retrieve records on disk using file processing and indexing techniques. Due to the uniform interface between the components and modules of the run-time sys-

tem, it will be quite straightforward to implement extensions to the algebra, the set of processing algorithms, or the storage structures.

6. Summary and Conclusions

In this prospectus, we have outlined a new approach to query optimization in behavioral object-oriented database systems, and have given a preliminary database architecture overview. The principal goal is to build an object-oriented database system with high performance for accessing large volumes of data. The fundamental problem we are addressing is how to utilize set-oriented processing and query optimization techniques without violating the object-oriented programming paradigm. Objects and classes can reveal execution information to the optimizer prior to execution, thus allowing object-at-a-time algorithms to be replaced with more efficient set-at-a-time algorithms. A single message representing a query can be transformed into a an execution tree with messages or file access at the leaves. Such query execution trees can be optimized using an optimizer based on the EXODUS optimizer generator. Query evaluation will be performed by three interacting software components, namely the object processor, the set processor, and the file manager. The set processor consists of an extensible set of algebraic operator implementations. An efficient, standardized interface between these operators allows augmentation of the algebra of object sets if needed.

Using this new database architecture, we hope to be able to provide the power of behavioral object-oriented data models and, by employing a query optimizer, to process queries on large sets of objects efficiently.

Acknowledgements

David DeWitt made the EXODUS optimizer generator possible, and challenged the first author to design a query optimization component for object-oriented database systems.

References

Batory1986a.
D.S. Batory, "GENESIS: A Project to Develop an Extensible Database Management System," *Proceedings of the Int'l Workshop on Object-Oriented Database Systems*, pp. 207-208 (September 1986).

Carey1985a.
M.J. Carey and D.J. DeWitt, "Extensible Database Systems," *Proceedings of the Islamorada Workshop*, (February 1985).

Carey1986a.
M.J. Carey, D.J. DeWitt, D. Frank, G. Graefe, J.E. Richardson, E.J. Shekita, and M. Muralikrishna, "The Architecture of the EXODUS Extensible DBMS: A Preliminary Report," *Proceedings of the Int'l Workshop on Object-Oriented Database Systems*, pp. 52-65 (September 1986).

Copeland1984a.
G. Copeland and D. Maier, "Making Smalltalk a Database System," *Proceedings of the ACM SIGMOD Conference*, pp. 316-325 (June 1984).

Freytag1987a.
J.C. Freytag, "A Rule-Based View of Query Optimization," *Proceedings of the ACM SIGMOD Conference*, pp. 172-180 (May 1987).

Graefe1987a.
G. Graefe, "Rule-Based Query Optimization in Extensible Database Systems," *Ph.D. Thesis*, University of Wisconsin, (August 1987).

Graefe1987b.
G. Graefe and D.J. DeWitt, "The EXODUS Optimizer Generator," *Proceedings of the ACM SIGMOD Conference*, pp. 160-171 (May 1987).

Graefe1987c.
G. Graefe, "Software Modularization with the EXODUS Optimizer Generator," *IEEE Database Engineering*, (December 1987).

Lohman1988a.
G.M. Lohman, "Grammar-Like Functional Rules for Representing Query Optimization Alternatives," *Proceedings of the ACM SIGMOD Conference*, pp. 18-27 (June 1988).

Maier1986a.
D. Maier, J. Stein, A. Otis, and A. Purdy, "Development of an Object-Oriented DBMS," *Proceedings of the ACM Conference on Object-Oriented Programming Systems*, (September-October 1986).

Maier1986b.
D. Maier, "Why Object-Oriented Databases Can Succeed Where Others Have Failed," *Proceedings of the Int'l Workshop on Object-Oriented Database Systems*, p. 227 (September 1986).

Manola1986a.
F. Manola and U. Dayal, "PDM: An Object-Oriented Data Model," *Proceedings of the Int'l Workshop on Object-Oriented Database Systems*, pp. 17-25 (September 1986).

Rowe1987a.
L. Rowe and M. Stonebraker, "The POSTGRES Data Model," *Proceeding of the Conference on Very Large Databases*, pp. 83-96 (August 1987).

Schwarz1986a.
P. Schwarz, W. Chang, J.C. Freytag, G. Lohman, J. McPherson, C. Mohan, and H. Pirahesh, "Extensibility in the Starburst Database System," *Proceedings of the Int'l Workshop on Object-Oriented Database Systems*, pp. 85-92 (September 1986).

Skarra1986a.
A.H. Skarra, Z.B. Zdonik, and S.P. Reiss, "An Object Server for an Object-Oriented Database System," *Proceedings of the Int'l Workshop on Object-Oriented Database Systems*, pp. 196-204 (September 1986).

Stonebraker1986a.
M. Stonebraker and L.A. Rowe, "The Design of POSTGRES," *Proceedings of the ACM SIGMOD Conference*, pp. 340-355 (May 1986).