PREDATOR : An OR-DBMS with Enhanced Data Types

Praveen Seshadri
Computer Science Department
Cornell University, Ithaca, NY
praveen@cs.cornell.edu

Mark Paskin
Computer Science Department
Cornell University, Ithaca, NY
paskin@cs.cornell.edu

1 Introduction

We are witnessing an explosion in the volume and complexity of digital information that people want to access and analyze. Much of this data is "multi-media" like images, video, audio, and documents. Several other kinds of complex and semi-structured data are also important, including geographical objects, chemical and biological structures, mathematical entities like matrices and equations, and financial data like time-series. Database systems must efficiently support queries over such richly structured data; otherwise, they will fast become "roadkill on the information super-highway" [DeW95].

PREDATOR is an object-relational DBMS being developed at Cornell University. It uses a novel "Enhanced ADT" (E-ADT) technology to meet this challenge. E-ADTs add semantics to the complex data types, resulting in dramatically improved performance, while retaining the extensibility of the database system. The demonstration shows the following features of PREDATOR: (a) standard relational query processing capability through SQL queries, (b) extensibility with complex data types (images, audio, video, documents), (c) content-based feature extraction, indexing and retrieval using path indexes, (d) extensible optimization of E-ADT expressions by reordering and merging, (e) WWW/Java- based user interface that supports extensible querying and display mechanisms.

These capabilities are demonstrated individually, as well as in the context of the following applications : (1) a multimedia digital library application involving thousands of images from museums of rare art, (2) a GIS application demonstrating raster, point and polygon data, and multi-dimensional indexing.

The PREDATOR system will be publicly available in summer 1997, along with source code. The goal of the release of this software is for it to be widely used for research and educational purposes. We will provide extensive web-based documentation of both the usage and the internal design details. No other freely available research system provides these advanced features and capabilities; consequently, we hope that PREDATOR will be used as a test-bed for a variety of research ideas.

SIGMOD '97 AZ,USA

Section 2 provides a brief technical background for the research ideas in PREDATOR, and a quick description of the PREDATOR system. Section 3 presents the details of the demonstration.

2 Background

The existing support for complex data in object-relational database systems (OR-DBMSs) is based on Abstract Data Types. ADTs were adapted from programming language concepts [LZ74] to databases in the 1980s [SRG83, Sto86]. Prominent OR-DBMSs based on ADTs include Illustra [Ill94], Postgres [SRH90] and Paradise [DKL+94]. The OR-DBMS maintains a table of ADTs and new ADTs may be added by a database developer or user. Each ADT provides a number of primitive operations for manipulating or querying values of the type. An ADT for images might provide operations *Rotate(I, Angle)*, *Clip(I, Region)*, and *Overlay(I1, I2)*. These primitive operations are composed to form query expressions like *Overlay(Rotate(I1, 90), Clip(I2, {0, 0, 100, 200}))*, that can be embedded within an SQL query. This is the current state of the art. The cost of ADT operations often dominates the overall execution cost of a query.

2.1 Possible Optimizations

Consider an OR-DBMS based on the ADT approach, and assume that an image ADT has been added. While image data is stored on disk in a compressed format, the ADT operations are implemented on an uncompressed main-memory image data structure. Consequently, the image argument of any operation is converted to its main-memory uncompressed form before an operation is invoked on it.

An earth scientist maintains a table of geographic data, each entry having a satellite photograph and several other columns. An SQL query asks for a rotated portion of each photograph of the arctic region. The cost of this query is dominated by the operations on the images.

```
SELECT Clip(Rotate(G.Photo, 90), {0, 0, 100, 200})
FROM   GeoData G
WHERE G.Region = 'arctic'
```

How is this query evaluated in current OR-DBMSs? For every data entry corresponding to the arctic region, the Photo attribute is retrieved from disk and decompressed into a main-memory image. The *Rotate* operation is then applied to it, and the resulting image is written to a compressed disk-resident form. The *Clip* operation is then applied with

the intermediate result image as its input. This input image is decompressed to a main-memory form, it is clipped to the desired dimensions, and the resulting image is written out to disk. One could improve this execution strategy as follows:

- It is unnecessary for *Rotate* to compress and write its result to disk. Instead, it could be passed directly in memory to *Clip*.
- *Rotate* is an expensive operation, whose cost depends on the size of the image being rotated. It would be cheaper to evaluate the equivalent expression *Rotate(Clip(G.Photo, {0, 0, 200, 100}), 90)*. By performing *Clip* early, there can be significant reductions in the cost of *Rotate*. Note that the arguments of *Clip* have changed as a result of its being applied before *Rotate*.
- If *Clip* is applied before *Rotate*, the entire image does not need to be retrieved from disk. Only the appropriate portion of the image is needed.

A combination of these strategies can lead to performance improvements of an order of magnitude. Essentially, an entire ADT expression needs to be treated *declaratively* and optimized. The textual representation of an expression should not specify an evaluation plan. This is the basic principle supported by E-ADTs in PREDATOR. Current OR-DBMSs based on the ADT approach violate this principle; they do not perform such optimizations. Consequently, their performance is poor. The basic research idea in PREDATOR is to provide mechanisms for each data type to specify the semantics of its methods; these semantics are then used for query optimization. In [SLR97], we presented various categories of optimization techniques for E-ADT expressions.

2.2 What is PREDATOR?

PREDATOR[1] is a client-server OR-DBMS. The core of the system is an extensible table in which E-ADTs are registered. Several E-ADTs have been implemented, including images, audio, video, documents, polygons, and sequences. A complex object like an image or a document can be a field within a relational tuple. The multi-threaded server is built on top of a layer of common database utilities that all E-ADTs can use. An important component of the utility layer is the SHORE Storage Manager [CDF+94] library, which provides facilities for persistent storage, concurrency control, recovery and buffer management.

An important feature of the system design is that E-ADTs are totally modular, with all E-ADTs presenting an identical internal interface. Relations are also implemented as an E-ADT. There are several interesting design issues that arise in building such a system, especially since we plan to optimize the E-ADT expressions — we discuss them in [SLR97].

3 Content of Demonstration

The aim of this demonstration is to show the effects of the research ideas in PREDATOR, and to highlight the other features that will make it attractive as a research and educational vehicle. While these features and ideas can be demonstrated in a stand-alone fashion, we have also built applications that place these demonstrations in a real-world context.

[1] PREDATOR stands for the PRedator Enhanced DAta Type Object Relational DBMS

1. *Basic Relational Functionality:* PREDATOR can support standard SQL functionality including INSERT and DELETE statements, and complex queries. A cost-based relational query optimizer using dynamic programming [SAC+79] is present. Further, there is a query-rewrite engine for heuristic rewrites (a simpler version of the query rewrite mechanism built in Starburst [PHH92] and available for the first time in a freely available DBMS). We show this functionality by executing some queries from the following benchmarks:
   - Wisconsin benchmark : the entire Wisconsin benchmark runs, and the query optimizer makes it run efficiently.
   - TPC-D benchmark : the complex queries of this benchmark involve several advanced SQL features. In order to process them efficiently, query rewrite is crucial.

   While this is not an primary component of the actual demo, it is important to demonstrate this functionality to show the extent to which the system has been developed.
2. *Basic Object-Relational Functionality:* The basic ability to add new data types is demonstrated. This requires an extensible type system, and extensible query processing. Each of the following features is demonstrated independently or through one of the applications described later:
   - Complex Data Types : images, audio, video, documents, points, polygons, rasters, molecules, sequences in a value-ADT model.
   - Methods on Complex Data Types : each data type can specify methods for manipulation and querying of the objects.
   - Indexes on Path/Function Expressions : indexes can be built on path/function expressions, and can be used for retrieval
   - E-ADT "Wizard": a toolkit is available to aid with the process of data type development.
3. *Java/WWW -based GUI* : This GUI allows a web-browser to act directly as a database client! How does it work? The entire GUI is a Java applet downloaded from the database server. The applet can customize the display based on user input. Further, new data types can be added seamlessly to the server, and they get displayed and queried appropriately at the client.
   - E-ADT specific display methods : if one field of the result of a query is an image, how should it be displayed? Image is an E-ADT, so the display routines cannot be hard-coded into the GUI. Instead, an extensible mechanism allows each new data type to register a Java class that handles the display of its objects at the client. We demonstrate this feature in use.
   - E-ADT specific query interface : if we wish to construct a query with an expression involving images, how does one know what methods are supported on images? More generally, can each E-ADT define its "query functionality" in an extensible graphical manner? The answer is yes, and we demonstrate this.

- Multiple interfaces/query languages : several different interfaces can be supported by the same database server, and new interfaces can be added without large scale modifications to existing GUI code. We show an SQL interface, as well two application-specific interfaces.

4. *Application 1: GIS* : a Geographic Information System, built as a specific interface on top of the GUI described above. This system uses points, polygons and raster images, with data from the Sequoia benchmark [SFGM93]. The purpose of this application is to show the effects of the E-ADT optimizations functioning underneath the extensible GUI.

   - Sequoia Benchmark: we show all the basic Sequoia benchmark queries running efficiently (except the transitive closure query, which is not supported)
   - Multi-dimensional indexing : R* -trees are built on the multi-dimensional data and result in performance improvements. Further, there are E-ADT optimizations that can be performed during index creation and index matching (for R*-trees and B+-trees). Both kinds of optimizations are demonstrated.
   - Optimization involving E-ADT aggregates: when there are aggregates involving E-ADTs (as in the Sequoia benchmark), there is further opportunity for optimizations. This is demonstrated in the context of multi-resolution storage of raster images.

5. *Application 2: Multi-Media Digital Library:* The second application implements a digital library, composed of several different types of multi-media data. An important component is a collection of rare art images from several national museums.

   - This is a real-life data set, and its size shows the robustness of PREDATOR
   - E-ADT optimizations : several categories of E-ADT optimizations are demonstrated, including the important optimizations of reordering and merging E-ADT methods
   - Feature extraction: several distinct image feature extraction methods are implemented. We demonstrate their use in similarity queries with efficient indexed retrieval.
   - Multi-resolution storage: the images are stored at multiple resolution levels. We show how the E-ADT query optimization utilizes the multi-resolution storage information for efficient query processing.

4 Future Directions

The code base of the system (currently about 50,000 lines of C++ code) will be made freely available in summer 1997. This will include several complex data types. Much of our current development effort is in "bullet-proof"ing the system for the release. We intend to actively support the code-base after the release and to use it as a research vehicle. More importantly, we hope that others might use this as a research vehicle too, given that it has extensive functionality. This might provide a large portion of the database research community with a common substrate on which to collaborate and compare efforts.

We have submitted a paper to VLDB-97 describing E-ADTs, and using performance numbers from PREDATOR. Another paper describing advanced optimization techniques for E-ADTs is in preparation. A paper on support for sequence data published in VLDB-96 [SLR96] used performance numbers generated from an earlier version of PREDATOR. The system has also been used in a Fall 96 advanced database systems course at Cornell as a research vehicle for student projects. Our current research is focused primarily on improving the mechanisms by which E-ADTs are supported and optimized. Specific issues include the use of semantics in indexing, optimization across E-ADT boundaries, the interaction between E-ADT expressions and relational query processing, and the extension of E-ADT concepts to heterogeneous query processing. We also hope to soon explore data mining techniques over complex data types.

5 Conclusion

For reasons of brevity, we do not describe related work; database systems like Illustra [Ill94], Postgres [SRH90] and Paradise [DKL+94] would be considered related to PREDATOR. A discussion of the differences between PREDATOR and these other systems is presented in [SLR97]. A brief summary of that discussion is: PREDATOR allows the semantics of complex data types to be incorporated into query optimization and query processing. Consequently, the performance of PREDATOR is considerably better than any of the other systems.

This demonstration shows the capabilities of PREDATOR and the underlying E-ADT paradigm. We believe that the next-generation of object-relational database systems should be based on E-ADTs. PREDATOR is a system that can act as the test-bed for research ideas that lead to this next-generation of database systems — we hope to facilitate this by making the system and the source code publicly available.

Acknowledgments

The implementation of PREDATOR has been greatly aided by Fabian Camargo, Dave Koster, Anil Sachdeva, Sandeep Tamhankar, Ed Chao and Leong Kian Fai. All of them have implemented pieces of functionality that went into this demonstration. The initial E-ADT ideas were developed at U.Wisconsin along with Raghu Ramakrishnan and Miron Livny. We also thank the SHORE development team at Wisconsin for the underlying storage management software.

References

[CDF+94] M.J. Carey, D.J. DeWitt, M.J. Franklin, N.E. Hall, M. McAuliffe, J.F. Naughton, D.T. Schuh, M.H. Solomon, C.K. Tan, O. Tsatalos, S. White, and M.J. Zwilling. Shoring up persistent objects. In *Proceedings of ACM SIGMOD '94 International Conference on Management of Data, Minneapolis, MN*, pages 526–541, 1994.

[DeW95] David J. DeWitt. DBMS: Roadkill on the Information Superhighway? Invited Talk: VLDB 95, 1995.

[DKL+94] D.J. DeWitt, N. Kabra, J. Luo, J.M. Patel, and J. Yu. Client-Server Paradise. In *Proceedings of the Twentieth International Conference on Very Large Databases (VLDB), Santiago, Chile*, September 1994.

[Ill94] Illustra Information Technologies, Inc, 1111 Broadway, Suite 2000, Oakland, CA 94607. *Illustra User's Guide*, June 1994.

[LZ74] B. Liskov and S. Zilles. Programming with Abstract Data Types. In *SIGPLAN Notices*, April 1974.

[PHH92] Hamid Pirahesh, Joseph Hellerstein, and Waqar Hasan. Extensible/Rule Based Query Rewrite Optimization in Starburst. In *Proceedings of ACM SIGMOD '92 International Conference on Management of Data, San Diego, CA*, 1992.

[SAC+79] Patricia G. Selinger, M. Astrahan, D. Chamberlin, Raymond Lorie, and T. Price. Access Path Selection in a Relational Database Management System. In *Proceedings of ACM SIGMOD '79 International Conference on Management of Data*, pages 23–34, 1979.

[SFGM93] Michael Stonebraker, James Frew, Kenn Gardels, and Jeff Meredith. The Sequoia 2000 Storage Benchmark. In *Proceedings of ACM SIGMOD '93 International Conference on Management of Data, Washington, DC*, pages 2–11, 1993.

[SLR96] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. The Design and Implementation of a Sequence Database System. In *Proceedings of the Twenty Second International Conference on Very Large Databases (VLDB), Bombay, India*, pages 99–110, September 1996.

[SLR97] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. The Case for Enhanced Abstract Data Types. Technical Report TR-97-1619, Cornell University, Computer Science Department, February 1997.

[SRG83] M. Stonebraker, B. Rubenstein, and A. Guttman. Application of Abstract Data Types and Abstract Indices to CAD Data Bases. In *Proceedings of the Engineering Applications Stream of Database Week*, San Jose, CA, May 1983.

[SRH90] Michael Stonebraker, Lawrence Rowe, and Michael Hirohama. The Implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, March 1990.

[Sto86] Michael Stonebraker. Inclusion of New Types in Relational Data Base Systems. In *Proceedings of the Second IEEE Conference on Data Engineering*, pages 262–269, 1986.