1996

A Formal Preparation for Object-Oriented Query Optimisation

Catherine Higgins
Technological University Dublin, catherine.higgins@tudublin.ie

Recommended Citation

Higgins C. (1996) A Formal Preparation for Object-Oriented Query Optimisation. In: Murphy J., Stone B. (eds) OOIS' 95. Springer, London. doi:10.1007/978-1-4471-1009-5_8

A Formal Preparation for Object-Oriented Query Optimisation

Catherine Higgins

Department of Maths and Computer Science,
Dublin Institute of Technology,
Kevin Street,
Dublin 8.

E-Mail : chiggins@dit.ie Fax : 353-1-4780282

ABSTRACT: *This paper describes work that is in progress on a formalised preparation to object-oriented query optimisation. Such preparation is conducive to the development of optimisation strategies. As an example of a formal preparation, this paper presents a formalised object algebra, a suggested optimisation method and an implementation of an algebraic converter suitable for DAPLEX.*

KEY WORDS: *Query Optimisation, Object-Oriented Databases, Object Algebra, Formal Approach.*

1. Introduction

An efficient method of processing database queries is a necessary feature in a successful database system. Traditionally, query optimisation has been the forte of relational systems. However, it is necessary that object-oriented database systems yield at least the same performance level that relational systems can achieve; hence query optimisation is also a necessary feature of object-oriented systems.

One way of ensuring that well-defined optimisation strategies can be developed for an object-oriented data model is to ensure that the model has a formally defined object algebra. Once it has been shown that the algebra has at least the same expressive power as the query language of the data model, then the algebra can be used in the optimisation process. Criteria for developing object algebras have been developed by Yu and Osborn [Yu90] and as these criteria complement Atkinson's manifesto - which states 13 tenets describing mandatory features of object-oriented databases [Atkinson90] - they were used as a guide to the development of the algebra for this paper.

The functional data model and DAPLEX [Shipman81] have been used as a basis for some object-oriented models and languages e.g. IRIS [Fishman87], PDM [Manola86] and P/FDM [Paton90], [Jiao91], [Gray92]. P/FDM (an integration of Prolog with FDM) has an optimiser which is based on the equivalence of DAPLEX queries to ZF-expressions [Turner81]. The object algebra described in this paper is based on the P/FDM version of DAPLEX.

The remaining structure of the paper is as follows - section 2 describes why a formal approach was taken as a precursor to query optimisation. Section 3 briefly describes DAPLEX and presents a selection of operators from the object algebra. Sections 4 describes the implementation of a parser and lexical analyser which convert DAPLEX queries into their algebraic equivalent. Section 5 suggests a method of query optimisation that could be suitably employed when optimising the algebraic form of queries and section 6 concludes this paper.

2. Why a Formal Approach to Query Optimisation?

To date, no formal model of query optimisation has been agreed on for object-oriented database systems. Therefore, in the absence of such a model, it is important that the optimisation strategies developed for individual object-oriented data models are based on formal foundations. If formal, preparatory work is carried out on a data model prior to developing optimisation strategies, then such strategies can be developed in a structured, well-defined environment. It is imperative that any proposed method of optimisation is formally defined so that it can be proved that the transformation of a query into a more optimal form does not change the semantics. One way of preserving the semantics of a query is to base the optimisation on an object algebra. This algebraic approach supports the ability to pose queries that are independent of physical structures and provides a structured format for query optimisation. Once an algebra has been formulated, it must be shown that it is semantically equivalent to the query language. This can be accomplished by formally specifying the semantics of both representations. Note that a presentation of these semantics is outside the scope of this paper. Since the formalised operators are high-level and symbolic, they can be manipulated with a variety of equivalence preserving transformation rules and heuristics to produce a more optimal version of the query. Examples of existing object algebras are PDM algebra [Manola86] and Shaw and Zdonik's algebra for the ENCORE data model [Shaw89].

3. Object Algebra for DAPLEX

The basic concepts in DAPLEX are entity classes which model real-world objects within the database and functions which represent entity attributes, relationships between entities and operations on entities. Entity classes can be arranged into a hierarchy that demonstrates property and behaviour inheritance between superclasses and subclasses. The format of a typical DAPLEX query is as follows :

for each *E1* in *entity_1* such that *predicate1*
  for each *E2* in *entity_2* such that *predicate2*
    etc.
      Actions;

where each *Ei* is a variable successively bound to instances of *entity_i* and *predicate_i* is a Boolean expression which may involve functions and quantifiers. *Actions* such as updating or displaying the retrieved objects can then be performed. This style of query is called navigational as it follows pointers (object identifiers) from entity to entity via function calls.

The object algebra developed for DAPLEX operates on classes, on individual objects, on sets of objects and on functions. A class is represented as a set of identifiers which represent the objects and subclasses of the class. Therefore, the semantics of a class are equivalent to that of a set. The term *set* in this paper is considered to be the DAPLEX definition of a set; i.e. a set can represent a class, it can be defined as the result of a multi-valued function, it can consist of a sub-range of ordinal values or it can be the result of applying set operators to existing sets.

In the algebraic operators to follow, each operator is declared with one or more arguments. The input arguments are preceded with a '+' (plus) sign and output arguments with a '-' (minus) sign. Optional arguments are enclosed in [ ]. A set of values is indicated by { }. A description of a sample selection of operators now follows.

Select (+Set, +[Predicate], -{Id})
Select retrieves the elements of a set. The *predicate* expression is optional and can represent a combination of functions and/or values connected together by relational and/or logical operators.

If the set represents a class then its elements are object identifiers and class identifiers. The class identifiers act as pointers to subclasses.

Select_direct (+Set, +[Predicate], -Id)
This operator allows direct access to a specific element in a set. If there is no predicate, it is assumed that the set is a singleton otherwise the predicate is used to identify and retrieve a specific object from the set. If the set represents a class, the predicate could represent a key. P/FDM supports the concept of a key which is an external identifier. It is used for directly accessing an object and for checking duplicate entries.

Apply_function (+funct_name, +Oid)
Functions are used in DAPLEX to represent attributes, relationships between objects and operations performed on objects. One operator is used to enforce these three uses of functions as from an implementation point of view the three concepts are very similar. *Apply_function* is also used for navigation to deal with both aggregation and inheritance hierarchies.

Update_multi_funct(+Function_name, {+Id}, +Set)
This is a general operator used for updating a multi-valued function by initialising / updating the return values of the function or by extending / restricting the function's return values. The *Id* parameter in this operator can represent an object, a class or an instance variable. The *Set* parameter represents the new or updated values that will be included or excluded from the return result of the multi-valued function.

Output_results (+Expression)
This can be viewed as a form of the relational algebra *Project* operator where the values of certain attributes are projected out. To avoid violating encapsulation, operations are used to factor out the required values. An expression can consist of functions and/or nested algebraic operators which evaluate to literal values.

Other algebraic operators include :-
Quantify_Class(+Class, +Oid), Delete(+Expression), Union (+Set1, +Set2, -NewSet),
Member_of (+Singleton, Set, -Boolean), Subclass_of (+Singleton, +Set, -Boolean).
Total(+Set, +Singleton, -Id) Average(+Set, +Singleton, -Id),
Identity_test(+Oid1, +Oid2, -Boolean), Deep_Equality(+Oid1, +Oid2, -Boolean),
Copy_ObjectInstance(+Oid, - NewOid)
Update_single_funct(+Function_name, {+Id}, +Singleton)
Maximum(+Set, -Id), Minimum(+Set, -Id), Count(+Set, -Id),
CreateActionGroup(+Name, +[{Parameters}], +{Operators}),

4. Implementation of lexical analyser and parser

The first step in the development of an optimiser for DAPLEX (that is based on the algebra) is the implementation of a lexical analyser and parser which will parse DAPLEX queries into their equivalent algebraic form. Turbo Prolog version 2.0 [Borland88] was chosen as the implementation language as it provides parser generator software.

A lexical analyser - which is used to convert DAPLEX queries into a series of tokens that can be read by the parser - was the first part of the converter to be implemented. Once the analyser was implemented then the parser was generated. The precursor to this generation was to write a grammar for DAPLEX using a BNF-like form. Once the grammar was formulated, the Turbo Prolog parser generator generated a parser which is used to convert DAPLEX queries into their algebraic equivalent. The output of the parser is in list form. Therefore, an optimiser can operate on the list by using rewrite rules and heuristics to generate a more optimal version of the

query before it is executed. An example of a DAPLEX query and its algebraic equivalent which is output from the parser is as follows :-

DAPLEX

```
for the b in book such that isbn_of(b) = "12345"
  print name_of(b);
```

Algebraic equivalent

```
[select_direct(id("b"),id("book"),
    equivalent(apply_function(id("isbn_of"),[id("b")]),str("12345"))),
    output_results(apply_function(id("name_of"),[id("b")]))]
```

5. Suggested Method of Optimisation of the Object Algebra

The proposed method of optimisation consists of two parts - a semantic and syntactic optimiser.

The semantic optimiser [Shenoy87], [Shenoy89] involves incorporating semantic knowledge into the optimisation process by using integrity constraints. At this stage, a query may be rejected as being semantically invalid and so would not be subjected to the syntactic optimisation process.

Transformation rules coupled with heuristics form the syntactic optimiser. Rule-based optimisation provides a clear, formal method of optimisation as the rules unambiguously describe the optimisation process in an implementation independent way. Rules can be formulated in terms of the algebra and can be derived from the areas of relational optimisation theory, P/FDM theory and object-oriented optimisation theory. An example of a transformation rule that advises the moving of restrictions so that they come as early as possible is as follows :

Select (+Set1, -Id1), Select (+Set2, +Pred_on_Set1_and_Set2, - Id2) =
Select (+Set1, +Pred_on_Set1, - Id1), Select (+Set1, +Pred_on_Set2, - Id2)

The steps involved in the proposed optimisation of DAPLEX queries are as follows :

1. The query is parsed into its algebraic equivalent.
2. The algebraic version of the query is subjected to the semantic stage of optimisation.
3. If the query is not rejected after stage 2, it is then subjected to the syntactic transformation rules. As a result of the output of this stage, different query evaluation plans can be generated. These plans depend on physical characteristics such as the existence of physical indexes, distribution of stored data and the clustering of data.

6. Conclusions

This working paper describes the formal preparations that were undertaken prior to developing an optimiser for DAPLEX. First an object algebra was generated for the query language, then this algebra was proven to be equivalent to the language by formally proving that their semantics are equivalent. Once the algebra was complete, a parser was written which converts DAPLEX queries into their algebraic equivalent. The paper concludes by suggesting a method of optimisation that is based on the object algebra. The main lesson learned from working on this research is that undertaking formal preparation to query optimisation can be as important as developing optimisation strategies.

References

[Atkinson90] Atkinson, Malcolm et al "The Object-Oriented Database System Manifesto" *Deductive and Object-Oriented Databases Ed: W. Kim, J. Nicolas, S. Nishio. Published by Elsevier Science 1990.*

[Borland88] Borland 1988 Turbo Prolog 2.0 User Guide, Turbo Prolog 2.0 Reference Guide, Turbo Prolog Toolbox.

[Fishman87] Fishman et al "IRIS : An Object-Oriented Database System" *ACM Transactions Office Information Systems Vol 5 No 1 Jan 1987.*

[Gray92] Gray, P.M.D. & Kulkarni K. & Paton N. W. *Object-Oriented Databases - A Semantic Data Model Approach. Published by Prentice Hall 1992.*

[Jiao91] Jiao, Z. & Gray, P.M.D. "Optimisation of methods in a Navigational Query Language" *Deductive and Object-Oriented Databases - Proceedings 2nd international conference. Ed : C. Delobel, M.Kifer, Y. Masunaga 1991.*

[Manola86] Manola, F.& Dayal U. "PDM: An Object-Oriented Data Model" *Readings in Object-Oriented Database Systems. Ed: S. Zdonik, D. Maier. Published by Morgan Kaufmann 1990.*

[Paton90] Paton, N. W. & Gray, P. M. D. "Optimising and Executing DAPLEX queries" *The Computer Journal, Vol 33, No 6 1990.*

[Shaw89] Shaw, G. & Zdonik, S. "An Object-Oriented Query Algebra" *Proc. 2nd International Workshop on Database Programming Languages 1989.*

[Shenoy87] Shenoy, S.T. & Ozsoyoglu, Z.M. "A system for Semantic Query Optimisation" *Proceedings 1987 SIGMOD conference May 1987.*

[Shenoy89] Shenoy, S.T. & Ozsoyoglu, Z.M. "Design and Implementation of a Semantic Query Optimiser" *IEEE Transactions on Knowledge and Data Engineering Sept 1989.*

[Shipman81] Shipman, D.W. "The Functional Model and the data language DAPLEX" *ACM Transactions on Database systems Vol. 6, No 1, March 1981.*

[Turner81] Turner, D. A. "The semantic elegance of applicative languages". *Proceedings ACM conference on Functional ProgrammingLanguages and Computer Architecture pp 81 - 100 1981.*

[Yu90] Yu, L. & Osborn, S. L. "An Evaluaton Framework for Algebraic Object-Oriented Query Models" *Report No. 275 Uni. Western Ontario, London, Ontario, Canada 1990.*