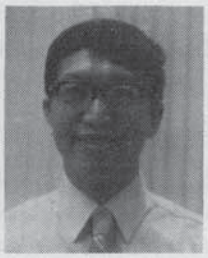
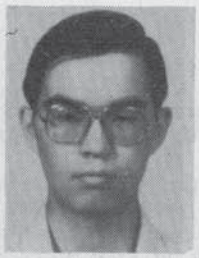


Member of IBM's Santa Teresa Laboratory during the summers of 1982 and 1983. He has published papers in switching and automata theory, computer graphics, operating systems, database, and most recently, software engineering and software metrics.

Stephen M. Thebaut (S'79-M'83) received the B.A. degree in mathematics from Duke University, Durham, NC, and the M.S. and Ph.D. degrees in computer science from Purdue University, West Lafayette, IN, in 1979 and 1983, respectively.

He joined the Department of Computer and Information Sciences, University of Florida, Gainesville, FL, as an Assistant Professor in 1983, and was supported by an IBM Postdoctoral Research Fellowship during the 1983-1984 academic year. His current research interests include software reliability, program maintenance, and the large-scale software development process.

Prof. Thebaut is a member of the Association for Computing Machinery, the IEEE Computer Society, and the 356 Registry.



Tze-Jie Yu received the B.S. degree in electrical engineering from National Taiwan University in 1979, and the M.S. degree in computer science from Purdue University, West Lafayette, IN, in 1983.

From 1981 to the present, he has been a Ph.D. candidate and Research Assistant at Purdue University. He worked for IBM's Santa Teresa Laboratory as a Systems Analyst during the summer of 1983. His research interests include software quality assessment, software tools, and software cost modeling.

Mr. Yu is a member of the Association for Computing Machinery and its Special Interest Group on Software Engineering, and a member of the IEEE Computer Society.

Lorri R. Paulsen received the B.S. degree in mathematics from Wagner College, Staten Island, NY, in 1968.

She joined IBM in 1968, and is currently a Program Development Manager for IBM General Products Division at Santa Teresa Laboratory, San Jose, CA. During her IBM career, she has been associated with systems design and development, working on the ASP Version 3 and JES3 products. In 1979 she became involved with the measurement of programmer's productivity, which led to her research with Software Science metrics and the prediction of error-prone modules. She is currently managing the development of software engineering tools to assist with the design and implementation of IBM products.

Translating SQL Into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries

STEFANO CERI AND GEORG GOTTLOB

Abstract—**In this paper, we present a translator from a relevant subset of SQL into relational algebra. The translation is syntax-directed, with translation rules associated with grammar productions; each production corresponds to a particular type of SQL subquery.**

The translation is performed in two steps, associated with two different grammars of SQL. The first step, driven by the larger grammar, transforms SQL queries into equivalent SQL queries that can be accepted by a restricted grammar. The second step transforms SQL queries accepted by the restricted grammar into expressions of relational algebra. This approach allows performing the second step, which is the most difficult one, on a restricted number of productions.

The translator can be used in conjunction with an optimizer which operates on expressions of relational algebra, thus taking advantage of a body of knowledge on the optimization of algebraic expressions. Moreover, the proposed approach indicates a methodology for the correct specification and fast implementation of new relational query languages. Finally, the translator defines the semantics of the SQL language, and can be used for the proof of equivalence of SQL queries which are syntactically different.

Index Terms—**Program translation, query equivalence, query languages, query optimization, relational algebra, relational database model, SQL.**

Manuscript received June 13, 1984; revised September 6, 1984. This work was supported in part by a grant from Data Base Informatica.

The authors are with the Dipartimento di Elettronica, Politecnico di Milano, I 20133 Milano, Italy.

I. Introduction

THE use of the relational model and languages is becoming more and more popular for the development of new database management systems. Formal languages, such as the relational calculus and algebra, have been proposed by Codd [1]

0098-5589/85/0400-0324$01.00

and later developed in the literature of database management systems [2], [3]. In particular, relational algebra can be used as a model for describing many approaches to query optimization [3]-[6], [16]-[19].

Although some earlier proposals for relational manipulation languages have been in fact based on relational algebra [7] or calculus [8], the trend in the design of relational languages has at some point moved toward a different direction; this was probably due to the success of the SQL query language [9], developed for the System R prototype [10], which has become *de facto* a standard used by many other systems [11]-[13].

SQL is an easy-to-use language with a number of high-level constructs. In general, small SQL queries are quite easy to understand; however, SQL allows writing very complex queries, whose meaning is quite obscure, and whose optimization is extremely hard [14]. Sometimes, two complex queries in SQL happen to have the same meaning, although they are very different from a syntactic point of view. In this paper, we propose a method for translating SQL queries into expressions of relational algebra. This method also suggests an approach to the design of new query languages.

SQL has a basic structure based on the query block or subquery:

"SELECT target_list FROM relation_list WHERE predicate".

If the predicate corresponds to a selection formula in the sense of Ullman [2], then a query block is easily mapped to a relational algebra expression. However, SQL allows several features which are not trivially mapped:

1) the nesting of several query blocks

2) the use of keywords such as: EXISTS, ALL, ANY, IN in the connection between query blocks

3) the use of variables and of predicates connecting query blocks indirectly

4) the use of set comparison operations between two query blocks.

5) the grouping of tuples of a relation into subsets and the evaluation of aggregate functions on each subset.

In this paper, we show how all these features can be mapped to expressions of relational algebra.

Relational algebra has been preferred to relational calculus as target of our translator for the following reasons:

1) It is more procedural, because algebraic expressions also give the order of application of operations in the computation of the query; thus, it is a more appropriate model for query optimization and system implementation.

2) Many existing approaches to query optimization and equivalence are (or can easily be) expressed in relational algebra.

3) Relational algebra uses relations (or sets) as operands, while calculus is based on tuple variables. For the optimization of queries, particularly with distributed environments or special-purpose database machines, set-oriented models are more appropriate than tuple-oriented models [19].

We have used standard relational algebra, with the only addition of a new operation for the evaluation of aggregate functions, which was required by the last feature in the above list.

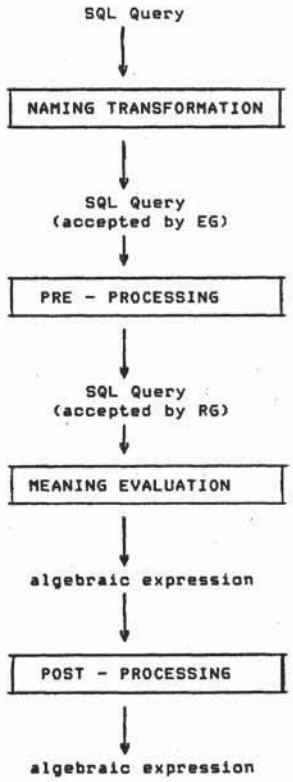


Fig. 1. Overall architecture of the translator.

A. Overview of the Proposed Approach

The overall architecture of the translator is shown in Fig. 1.

The translator is based on two grammars for the SQL language: an *extended grammar* EG, and a *restricted grammar* RG. EG does not accept all the statements accepted by the SQL grammar of [9], but it has approximately the same expressive power; it also includes some elements (such as the EXISTS and ALL keywords in the predicates) which were not present in the grammar of [9]. In the sequel, we will be more specific on the SQL statements which are not accepted by EG. The main difference between the grammar of [9] and EG is that EG allows a correspondence between types of SQL subqueries and productions of the grammar, thus allowing a syntax-directed translation [15]. RG has a smaller number of productions than EG, representing few types of SQL subqueries.

The queries which are accepted by the EG but not by the RG are *preprocessed* and transformed into queries which are accepted by the RG; thus, the preprocessing consists in a transformation from SQL into SQL. For the queries which are accepted by the RG, we have defined the *meaning* in relational algebra; thus, the meaning of a query consists in a translation from the query in SQL into an expression of relational algebra. Both the definitions of the preprocessing and of meaning are syntax-directed: rules are in correspondence with the productions of the EG and RG. The rationale for distinguishing the preprocessing from the meaning is that the meaning of an SQL subquery is, in general, rather difficult. Thus, we have determined a small set of subqueries which are representative of all other subqueries, and we have given the meaning just for them.