Chapter 12
Making Query Coding in SQL Easier by Implementing the SQL Divide Keyword:
An Experimental Query Rewriter in Java

Eric Draken
University of Calgary, Canada

Shang Gao
University of Calgary, Canada

Reda Alhajj
University of Calgary, Canada & Global University, Beirut, Lebanon

ABSTRACT
Relational Algebra (RA) and structured query language (SQL) are supposed to have a bijective relationship by having the same expressive power. That is, each operation in SQL can be mapped to one RA equivalent and vice versa. Actually, this is an essential fact because in commercial database management systems, every SQL query is translated into equivalent RA expression, which is optimized and executed to produce the required output. However, RA has an explicit relational division symbol (÷), whereas SQL does not have a corresponding explicit division keyword. Division is implemented using a combination of four core operations, namely cross product, difference, selection, and projection. In fact, to implement relational division in SQL requires convoluted queries with multiple nested select statements and set operations. Explicit division in relational algebra is possible when the divisor is static; however, a dynamic divisor forces the coding of the query to follow the explicit expression using the four core operators. On the other hand, SQL does not provide any flexibility for expressing division when the divisor is static. Thus, the work described in this chapter is intended to provide SQL expression equivalent to...
explicit relational algebra division (with static divisor). In other words, the goal is to implement a SQL query rewriter in Java which takes as input a divide grammar and rewrites it to an efficient query using current SQL keywords. The developed approach could be adapted as front-end or wrapper to existing SQL query system. Users will be able to express explicit division in SQL which will be translated into an equivalent expression that involves only the standard SQL keywords and structure. This will turn SQL into more attractive for specifying queries involving explicit division.

INTRODUCTION

Since its development as the standard query language for relational databases, the Structured Query Language (SQL) has witnessed a number of developments and extensions. Different research groups have worked on different aspects of SQL (Brantner et al., 2007; Chong et al., 2005; Harris & Shadbolt, 2005; Hung et al., 2005; Karvounarakis et al., 2002; Pan & Hein, 2003; Prudhommeaux, 2005). We realized the gap between relational algebra and SQL as far as the division operation is concerned. While the relational algebra has an operator for explicit division, SQL does not provide any keyword for explicit division. Hence from the user perspective the translation between SQL and the relational algebra is not one to one. A user who is able to express explicit division in relational algebra does not find the same flexibility with SQL. The work described in this paper is intended to cover this gap.

Given two tables S and T such that the schema of S is subset from the schema of T; a common type of database query requires finding all tuples of a table or view that are related to each and every one of the tuples of a second table or group, and is called Relational Division. For instance, it is very common to code queries for finding employees working on all projects; students who have completed a certain set of courses, etc. Such kind of queries require a division operator which is normally expressed internally as equivalent to a combination of four core relational algebra operators, namely, selection, projection, difference and cross-product.

The research community realized the importance of the division process and hence defined a standalone operator for explicitly expressing division as in the above two examples. However, the explicit division operator is not applicable when the divisor is dynamic. For instance, queries such as finding students who completed all first year courses in their departments, finding persons who ate at every restaurant in their neighborhood, etc, are not doable using explicit division; these queries qualify as requiring implicit division because the divisor changes for different instances of the dividend. Implicit division could be coded by explicitly using the four core operators, though expressing implicit division is not an easy task. Having no divide keyword for expressing explicit division in SQL, as many as six query types in different SQL dialects have been identified using nested select, exists, not exists, except, contains and count keywords (Elmasri & Navathe, 2006; McCann, 2003). We argue that it is necessary to provide for explicit division in SQL.

The division operator is firstly instantiated from the perspective of relational algebra (Dadashzadeh, 1989) and further extended to relational databases in terms of flexible implementation (Bosc et al., 1997). Further discussions shift the focus to advanced computation, such as Fuzzy systems (Galindo et al., 2001; Bosc & Pivert, 2006).

Once integrated into SQL, the explicit division will allow users to express their queries easier and hence one of the main targets of SQL (ease of use by end users) will be satisfied better. Realizing this
Related Content

Application of Mobile Agents in Mobile Data Access Systems: A Prototype
[www.igi-global.com/article/application-mobile-agents-mobile-data/3318?camid=4v1a](www.igi-global.com/article/application-mobile-agents-mobile-data/3318?camid=4v1a)

Data, Knowledge & Information in Database and Knowledge-Based Systems
[www.igi-global.com/article/data-knowledge-information-database-knowledge/51106?camid=4v1a](www.igi-global.com/article/data-knowledge-information-database-knowledge/51106?camid=4v1a)

The Expert’s Opinion
[www.igi-global.com/article/expert-opinion/51193?camid=4v1a](www.igi-global.com/article/expert-opinion/51193?camid=4v1a)

Querying and Integrating P2P Deductive Databases
[www.igi-global.com/chapter/querying-integrating-p2p-deductive-databases/20726?camid=4v1a](www.igi-global.com/chapter/querying-integrating-p2p-deductive-databases/20726?camid=4v1a)