Chapter 2
Set-Oriented Queries in SQL

Antonio Badia
University of Louisville, USA

ABSTRACT

Set-oriented queries are those that require a condition to hold of an arbitrary set of rows, either when comparing the set to a single row or to another set of rows. Examples of such queries are universally quantified queries and skyline queries. These queries, while important for data processing, are difficult to write and difficult to process. In this chapter we review research dealing with the specification and optimization of such queries. We present several approaches proposed to deal with universal quantification as well as with set predicates in general.

1. INTRODUCTION

Over the years, the SQL standard has evolved to adapt to new challenges. The initial core of the language has been extended to deal with object-relational extensions, XML data, and text data ((Melton, 2001a)). Also, new operators (like CUBE and WINDOWS) have been added to deal with Decision-Support style queries ((Melton, 2001b)).

An issue already present at the birth of the language is that SQL, like Relational Algebra (henceforth RA), is biased towards *row-based* queries. These are queries defined by conditions that require the existence of a row or a *fixed* set of rows with some given constraints. Such queries correspond to the SPJ (Select-Project-Join) subset of RA and the SELECT ... FROM ... WHERE *simple* fragment of SQL (with no subqueries and no set operators). By contrast, *set-oriented* queries are those that require a condition to hold of an *arbitrary* set of rows, either when comparing the set to a single row or to another set of rows. Examples of such queries are *universally quantified* queries and *skyline* queries. Such queries are expressed in RA with negation and in SQL with subqueries (especially those using negated operators, like NOT IN and NOT EXISTS), or set difference (EXCEPT). In many cases, these queries can be paraphrased using grouping and counting.

In this chapter, we overview research dealing with the processing and optimization of set-oriented queries. In the next section, we provide a definition of the class and examples to show both the importance of such queries and the challenge that they present to traditional query optimization. In section 3,

DOI: 10.4018/978-1-4666-8767-7.ch002

we overview early attempts to deal with set-oriented queries and some of the solutions proposed. Then, in section 4 we introduce the framework of *Generalized Quantification*, used in (Badia, 2009) to give a unified solution to the problem. After an overview of the basic theory, we show in section 5 how it is applied to support quantification in SQL, and provide examples and a summary of the results of this research, as well as a comparison with other approaches in the literature. Finally, in section 6 we discuss open issues and future avenues of research. We close with a brief recapitulation of the main points of the chapter in section 7.

2. MOTIVATION

Tuple-based conditions, both in RA and SQL, involve the value of one or more attributes in a given tuple, but they apply to one tuple at a time only. For instance, in the TPCH benchmark[1], we can write a query to get the suppliers from Germany with a condition country_name='GERMANY' applied to each tuple in the join of SUPPLIER and NATION. To write more complex conditions involving several tuples, RA requires that we 'line up' all the tuples by using joins. For instance, to get the suppliers that supply two parts, we need to see two tuples with the same supplier id and different parts ids. To achieve this, we join the relation SUPPLIES with itself and use renaming (say, giving SUPPLIES the alias S2), and then apply the condition SUPPLIES.supkey = S2.supkey and SUPPLIES.partkey <>S2.partkey[2]. Such cases, however, require a *fixed* amount of joins (one, in the example); they can be seen as conditions over a fixed number of tuples (two, in the example). There are queries, though, that require an indeterminate number of tuples. One example is the query *select the orders where all suppliers are from Germany*, in which we do not know in advance how many suppliers there are in each order, or how many suppliers are from Germany (and both sets can change with the data). These and similar examples (for instance, *select the orders where at least half of the suppliers are from Germany*) cannot be expressed with a regular SPJ query; they require set difference (i.e. negation) or other set operations. Queries containing such conditions are called *set-oriented queries*. The case of universal quantification ("all") has been known for a long time as problematic for SQL, both to express and to optimize.

Set-oriented conditions come in three main types in SQL: one is a simple condition in a single set -for instance, existential quantification (checking whether the set is empty or not) is expressed with EXISTS or NOT EXISTS. An example would be a query like

QUERY 1

```
SELECT c_name
FROM Customer C1
WHERE NOT EXISTS (SELECT c_name
FROM Customer
WHERE c_acctbal > C1.c_acctbal
and c_nation = C1.c_nation)
```

which retrieves the names of customers who have the largest account balance among the suppliers of a given nation. Note that this query has exactly the structure of a skyline query (Borzsony, Kossmann, & Stocker, 2001)[3].