Set Comparison in Relational Query Languages

Mohammad Dadashzadeh
Oakland University, USA

INTRODUCTION

Today’s de facto database standard, the relational database, was conceived in the late 1960’s by Edgar F. Codd at IBM. The relational data model offered the user a logical view of the data that was shielded from consideration of how the data would, in fact, be physically organized in storage. This feat was accomplished in large part by the introduction of relational query languages that would specify the desired set of records in a non-procedural fashion. In contrast to the prevailing record-at-a-time, loop-oriented, procedural query languages of the hierarchical and network database management systems, relational query languages were set-oriented in that they would operate on sets of records (i.e., relations or tables) at-a-time in order to produce the desired set of output records. Codd introduced both a relational algebra and a relational calculus as a basis for dealing with data in relational form. Indeed, he defined what the first relational language was: Data Sublanguage Alpha (Codd, 1971).

The non-procedural nature of relational query languages made it possible to envision that end users could be expected to formulate ad hoc queries without resorting to a programmer. To that end, RDBMS adoption was thought to be facilitated by creation of an English-like query language. The language created for this purpose at IBM was called SEQUEL (Structured English Query Language), though it eventually grew in scope to handle other tasks including database modification, definition, authorization, and transaction processing (Chamberlin et al., 1976). At about the same time, another IBM research group produced Query by Example (Zloof, 1975), which because of its graphical interface proved to be easier to use for casual users. However, the wider applicability of SEQUEL led to its adoption and standardization as SQL (Structured Query Language) between 1982 and 1986.

As a relational query language, SEQUEL borrowed features from both relational algebra and relational calculus. However, in an effort to appeal to end users, the expressive power of relational calculus quantification (universal, for all, and existential, there exists) was somewhat sacrificed in favor of algebraic grouping (Group By and SET operations). Unfortunately, the balanced approach of SEQUEL to relational calculus and relational algebra was abandoned in SQL, resulting in undue complexity when formulating queries requiring universal quantification. This article examines the shortcomings of relational query languages in formulating such set comparison queries and proposes solutions to overcome them with minimal effort.

BACKGROUND

Consider the following relational database about suppliers, parts, and jobs. (The primary key of each relation is underlined.)

SUPPLIER( S#, SName, Status, City )
PART( P#, PName, Color, City )
JOB( J#, JName, City )
SHIPMENT( S#, P#, J#, QTY )

The relation SHIPMENT records the quantity of each part being shipped by each supplier to various jobs. An instance of this database is depicted below.

Table 1. Supplier.db

Table 2. Part.db
Now, consider the following queries:

- **Q1**: List the suppliers who ship every red part. (Answer: S5)
- **Q2**: List the suppliers who do not ship to any job located in London. (Answer: S1 and S3)
- **Q3**: List the jobs that are only receiving parts warehoused in London. (Answer: None)
- **Q4**: List the suppliers who are shipping to exactly the same jobs as supplier S1. (Answer: None)
- **Q5**: List the suppliers who are shipping exactly the same parts to jobs located in London as they are shipping to jobs located in Athens. (Answer: S2 and S4)

Each of the above queries involves comparison of sets of values in two tables. For example, in Q1, the set of parts (P# values) associated with each supplier (distinct S# value) in the SHIPMENT table must be examined to determine if it contains the set of parts (P# values) in the PART table sharing the value of “Red” for the COLOR attribute.

Despite their innocuous appearances, queries involving set comparison that tend to arise frequently in online analytical processing (OLAP) situations are especially difficult to formulate in relational query languages (Blanning, 1993; Celko, 1997; Matos & Grasser, 2002; Rao, Badia, & Van Gucht, 1996). This article summarizes the existing approaches and the proposed solutions to set comparison queries in relational algebra, Query by Example, and SQL.

## SET COMPARISON IN RELATIONAL ALGEBRA

In relational algebra (Ramakrishnan & Gehrke, 2003), the DIVISION operator provides the mechanism by which a restricted form of set comparison may be directly formulated. To fix ideas, consider the following formulation of query Q1 in relational algebra.

**Q1: List the suppliers who ship every red part.**

Temp1 := **SELECTION** (PART) using Color = “Red”

Temp2 := **PROJECTION** (SHIPMENT) using S# and P#

Result := **DIVISION** (Temp2, Temp1) using Temp2.P# and Temp1.P#

Here, Temp1 is produced by performing the SELECTION operation on the PART table using the selection condition Color = “Red.” Next, the Temp2 intermediate table is formed by projecting columns S# and P# from
Related Content

Hong Zhang, Rajiv Kishore and Ram Ramesh (2007). Journal of Database Management (pp. 1-19).
www.igi-global.com/article/semantics-mibml-conceptual-modeling-grammar/3364?camid=4v1a

Relaxing Queries with Hierarchical Quantified Data Abstraction
www.igi-global.com/article/relaxing-queries-hierarchical-quantified-data/3394?camid=4v1a

Similarity Search in Time Series Databases
www.igi-global.com/chapter/similarity-search-time-series-databases/11218?camid=4v1a

Fuzzy Spatial Data Types for Spatial Uncertainty Management in Databases
www.igi-global.com/chapter/fuzzy-spatial-data-types-spatial/20365?camid=4v1a