Technical Perspective: Reflections on Extending SQL using Constraints

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Relational query languages enabled the programmer to express succinctly *what* data items to retrieve using a logical model of data without any knowledge of the underlying physical structures and helped relational systems gain widespread adoption. To support applications using a relational database effectively, there has been much work subsequently along the following three dimensions:

(a) Application developers needed a programmatic way to invoke relational query functionality from within their applications. The most primitive and most prevalent form of such integration uses ODBC or JDBC APIs. While they provide connectivity to database objects, the application programmer still must manage two separate type systems and programming models. LINQ (Language Integrated Query) is an elegant example of integration where query expressions are introduced as first class citizen in the programming languages. Object-relational mapping tools allow the application programmer to continue working in their object-oriented programming paradigm even though they may be storing and retrieving relational database objects.

(b) Enriching the relational model to support extensibility so that programmers could do more within a relational database system was another key direction that has been pursued. The simplest example of such extensibility was introduction of user-defined selection and user-defined aggregates which are widely supported in relational databases. Object-relational databases extended the relational model to support complex types, inheritance and support for user-defined methods but their adoption has been relatively modest. The extensibility mechanisms in relational databases have been especially useful in adding support for richer data types such as spatial.

(c) There have been many proposals to extend core SQL by making declarative querying in relational languages to do more. The language has been extended to add recursive queries, Roll-Up, Grouping Sets, Window function and more. In addition to the extensions that have been incorporated in the SQL Standard, over time there has been a steady stream of research proposals for enriching core SQL. Adoption of any such extension requires careful consideration as they directly impact complexity of the language and the query engine.

These directions of work are largely complementary. However, there is always a healthy tension between how much should be done in applications and what is best deferred to the database server using either its extensibility mechanisms or extensions to core SQL, i.e., (b) and (c) above. Issues that influence such a debate are ease of specification and efficient execution of desired computation, data movement, and increased complexity of the database platforms.

The following paper by Brucato, Abouized, and Meliou belongs to the line of work in (c) that suggests adding more functionality to core SQL. Their work builds on past research work in the broad area of endowing the query languages with also the power of specifying constraints. The motivation for adding constraint specification to SQL comes from the desire to marry the well-established paradigms of constrained optimization e.g., Integer Linear Programming (ILP), and traditional SQL querying. Of course, selection conditions in SQL are simple row level constraints. Past work on constraint query languages proposed more generalized constraints over row values. However, the following paper (as well as a few other recent papers) focuses on *aggregate constraints* that the set of answer rows to a query must satisfy collectively, and picks the answer set based on an objective criterion (analogous to the objective function in ILP). They give a nice motivating example of formulating a meal plan for which you want to ensure that the total number of calories of the chosen meal plan is within a range and the total fat consumption is minimized. The paper lays out specific extensions to SQL needed to declaratively capture such queries and explains how such queries can be evaluated by first executing the traditional relational queries and then mapping the constraint satisfaction and objective criterion to an ILP instance which in turn can be solved using any off-the-shelf ILP solver. It is indeed advantageous to use a well-tuned off-the-shelf ILP solver in the database server using its extensibility mechanism instead of adding complexity to the core SQL engine. The rest of the paper addresses the challenges in solving large ILP problems using offline partitioning and approximation techniques to break down the global ILP instance into smaller ILP sub-problems such that the off-the-shelf ILP solvers are able to handle the scale of each sub-problem. However, proposed techniques such as offline partitioning is subject to debate as partitioning criteria for data may depend on other workload on the system such as production queries.

While the paper is unlikely to end the debate on whether or not constraints should be added to SQL (since any addition to the SQL has complex trade-offs), the paper has studied what it takes to add constraints to SQL in a relatively complete way – language extension, impact on query execution, and techniques to cope with scale. If you are interested in the topic of constraint specification and optimization over information in databases, you should definitely pay attention to this paper. It is worth a read also for any researcher who wants to consider adding extensions to core SQL to ease application tasks as it illustrates the key considerations one must address.