Chapter VII
Translating Advanced Integrity Checking Technology to SQL
Hendrik Decker
Instituto Tecnológico de Informática, Spain

INTRODUCTION

The main goal of this chapter is to arrive at a coherent technology for deriving efficient SQL triggers from declarative specifications of arbitrary integrity constraints. The user may specify integrity constraints declaratively as closed queries in predicate calculus syntax (i.e., sentences in the language of first-order logic, abbr. FOL), as datalog denials, as query conditions in SQL WHERE clauses, or in some other, possibly more user-friendly manner (e.g., via a dialog-driven graphical or natural language interface which internally translates to equivalent WHERE clause conditions). As we are going to see, the triggers derived from such specifications behave such that whenever some update event would violate any of the integrity constraints, one or several of the triggers derived from that constraint are activated in order to enforce the constraint. That is, the violation is either prevented by rolling back the update or repaired instantly by subsequent further updates.

In this chapter, we describe how to implement advanced datalog technology for integrity checking in the framework of SQL. That is, we show how to represent and evaluate arbitrarily complex constraints in SQL without incurring major disadvantages usually associated to integrity checking in knowledge-rich applications. Error-prone procedural specification and laborious maintenance of integrity constraints is avoided by the declarativity of datalog. The cost of evaluation is considerably reduced by an automated
translation of declarative specifications to SQL triggers. That way, the advantages of declarativity of specification and efficiency of execution can be combined, while the performance disadvantage of CHECK clauses in ASSERTION statements, as well as the disadvantage of procedural specifications by the user, is avoided.

As already indicated, three different, though mutually related declarative languages for specifying integrity constraints are addressed in this chapter: FOL, datalog (i.e., Horn clauses with negation-as-failure) and SQL. Datalog is primarily of historical interest, but it fits well into our presentation since most of the techniques discussed in this chapter have been developed in the datalog framework of deductive database systems (cf. (Ramakrishnan & Ullman, 1995) for a survey). For representing arbitrarily complex integrity constraints in datalog, an extended first-order syntax is needed in the body of datalog queries. Thus, FOL is a natural choice for expressing integrity constraints.

In the first section, we survey the history and the state of the art of integrity constraint checking. In ‘Principles of Simplified Integrity Checking,’ we recapitulate common principles of simplifying integrity checking. In ‘An SQL Syntax for Integrity Constraints,’ we first define a syntax for integrity constraints as a subset of standard SQL. In ‘Translating Principles of Simplified Integrity Checking to SQL,’ we discuss the applicability of the principles in ‘Principles of Simplified Integrity Checking’ in an SQL framework. In ‘First-Order Logic Representation of Integrity Constraints,’ we discuss a FOL syntax for integrity constraints which is sufficiently expressive and lends itself well toward a straightforward translation into SQL. In ‘Translating Integrity Constraints to SQL Conditions,’ we describe a translation of constraints in this syntax to WHERE clause conditions. In ‘Identifying and Specializing Relevant Integrity Constraints,’ we describe how constraints represented as such conditions can be simplified for the purpose of improving the efficiency of integrity checking. In ‘Translating Integrity Constraints to Optimized SQL Triggers,’ we describe a translation of simplified SQL conditions into equivalent triggers. They closely correspond to what is called “update constraints” in (Decker, 1987). The syntactic transformations, rewrites and simplifications described in ‘First-Order Logic Representation of Integrity Constraints’ to ‘Translating Integrity Constraints to Optimized SQL Triggers’ are easily automated. In the conclusion, we summarize the chapter, address related work and point out directions for future work. For simplicity, we assume that updates are single tuple insertions or deletions in base tables. An extension to more general transactions does not pose essential new problems, but dealing with SQL transaction semantics (which are not yet
Object-Oriented Database Benchmarks
Jerome Darmont and Michel Schneider (2002). Advanced Topics in Database Research, Volume 1 (pp. 34-57).
www.igi-global.com/chapter/object-oriented-database-benchmarks/4321?camid=4v1a