

Why to Use Deductive Technology in SQL Databases?

Andreas Behrend
Universität Bonn, Institut für Informatik III
Römerstr. 164, 53117 Bonn
behrend@cs.uni-bonn.de

Abstract

The notion of a deductive database has emerged during the 1970s in order to describe database systems capable of inferring new knowledge using rules. Within this research area three kinds of rule-based extensions for traditional databases have been intensively studied: active, deductive and normative rules. Nowadays, even SQL databases use these extensions in form of triggers, views and integrity constraints, although with quite restricted functionality. A uniform approach making these concepts really orthogonal to each other is still missing. In this paper we will show how new results in the field of deductive databases may help to find a more uniform extension of SQL.

1 Deductive Databases

The notion of a deductive database has been used for systems capable of deriving new knowledge using deductive rules. These rules are usually expressed in the database query language Datalog because of its simplicity and readability. We will first use this language for presenting important database concepts and transfer methods and concepts from Datalog to SQL later. A Datalog rule consists of a single positive literal called head of the rule and a conjunction of positive or negative literals called body of the rule. Consider for example the following deductive rules

$$\begin{aligned} route(X, Y) &\leftarrow connection(X, Y), \neg blocked(X, Y) \\ route(X, Y) &\leftarrow connection(X, Z), route(Z, Y), \neg blocked(X, Z) \end{aligned}$$

specifying relation *route* as the transitive closure of all connections which are not blocked. The semantics of such deductive rules is defined by means of fixpoint operators which directly induce a set-oriented inference mechanism for determining derivable facts. A fixpoint approach is needed because of recursion in Datalog rules as in the example above. In extended Datalog, it is possible to define Boolean predicates as well, e.g.

$$\begin{aligned} cyclic &\leftarrow route(X, Y), route(Y, X) \\ two_connections &\leftarrow connection(X, Y), connection(Y, Z), X \neq Z \end{aligned}$$

where predicate *cyclic* is considered true if at least one route is cyclic while the second predicate *two_connections* is evaluated to true if at least two connected paths exists. Both predicates may be used as yes/no queries, e.g., in a rule body similar to the inequality test we used in the body of *two_connections*. In order to describe invariants against database state modifications, normative rules have been introduced which can be easily defined by means of Boolean predicates. Consider the following two normative rules

$$\begin{aligned} ! - two_connections & \qquad \qquad ! - \neg cyclic \end{aligned}$$

the first one specifying that in every database state two connective paths must exist, whereas the second constraint specifies that no cyclic routes may be derivable. A normative rule combined

with a set of deductive rules correspond most directly to the general perception of static integrity constraints as invariants.

As a third type of rule, active rules have been studied for specifying the automated execution of actions in response to specific events such as database updates. The combination of deductive and active rules is particularly desirable for designing more advanced applications which clearly separate declarative and imperative specifications [Man94]. For this purpose, the language ActLog has been developed in our group as an active extension of Datalog [Gri97]. As an example consider the following ActLog rules m_1 and m_2 which may be used for materializing the view *route*, introduced above:

$m_1 :$	$+materialize_route \Rightarrow$	Event
	$+m_route(X, Y) \leftarrow$	Action
	$connection(X, Y) \wedge \neg blocked(X, Y)$	Condition
$m_2 :$	$+m_route(Z, Y) \Rightarrow$	Event
	$+m_route(X, Y) \leftarrow$	Action
	$connection(X, Z) \wedge \neg blocked(X, Z)$	Condition

The rule m_1 is triggered by the 'artificial' event $+materialize_route$ used for initiating the materialization process. Afterwards, the materialized 'simple' routes in relation *m_route* trigger the second rule m_2 which successively inserts transitive connected routes into *m_route*. Note the similarities between deductive rules in Datalog and the Action/Condition part of ActLog rules. The action part of an ActLog rule corresponds to a rule head, whereas conditions directly correspond to a rule body allowing references to both derived and extensional relations. ActLog rules can be triggered by explicit and implicit updates, i.e., explicit insertions into and deletions from base relations as well as implicit changes of derived relations are possible event specifications.

Supporting these three kinds of rule-based extensions for traditional databases, we may expect the following base functionalities in such an active and deductive database. From the view of deductive database systems, an inference mechanism for evaluating deductive rules must be integrated which also allows for efficient recursive query processing, and which provides a mechanism for efficient static integrity checking. In addition, we may expect that materialized views are supported and general view updating is possible. From the view of active databases, it is beneficial to combine active and deductive rules in such a way that complex events may be implicitly defined by means of deductive rules [Man94]. This requires in particular the computation of induced updates, i.e. update propagation, which is also necessary for efficient integrity checking and materialized view maintenance. All of these different tasks have been already widely investigated on their own. However, it has been recently shown that it is possible to base them all on the same inference mechanism leading to a very simple system architecture which integrates these three kinds of rules and the corresponding functionality. A system which supports these functionalities allows the design of more advanced applications as needed in spatial databases, expert systems and mediators, for example.

2 SQL:1999 and SQL-Based Systems

Active, deductive and normative rules are also present in the actual standard SQL:1999 by means of triggers, views, constraints, and assertions each of them implemented to a certain extent in commercial systems. We will now look a bit closer at each rule concept in SQL and discuss their restrictions as well as the implications of a possible extension of SQL:1999 in the future. We begin with normative rules in SQL:1999 as one of the oldest rule concepts in this language which have been implemented in commercial products (such as, e.g., Oracle or DB2) in a very limited way so far.

2.1 Constraints in SQL

Rather than providing a single constraint concept, SQL offers four main syntactic categories of static constraints: assertions, table constraints, column constraints, and domain constraints. Assertions are the most general form of integrity constraint in SQL and correspond most directly to the concept of normative rules introduced above. However, assertions as well as table constraints are usually not implemented in commercial products probably due to efficiency reasons. This is justified to a certain extent, as one needs to provide an efficient mechanism for computing induced changes in order to restrict integrity checking to the affected part of a database only. On the other hand, update propagation methods have been widely studied in the context of deductive databases and the transfer of such approaches from a Datalog context to the SQL world is obviously possible in principle though intricate in detail.

Many authors have proposed techniques for incremental integrity checking generally leading to specialized versions of the original constraints which can be checked more efficiently. However, the only approaches using SQL were published by Ceri/Widom [CW90] in 1990 and by Pieper in [Pie01], the latter one being preferred as it truly relies on the new trigger concept in SQL:1999. It is, however, also possible to base update propagation on the view concept in SQL. In the next section we will briefly review the view concept in SQL:1999 and its implementation in commercial systems.

2.2 Views

Because of the recent extension in SQL:1999 which allows the definition of some kind of limited recursive views, this concept now corresponds rather directly to deductive rules in Datalog. In SQL:1999 all views are virtual, i.e., no materialized views can be defined. This again is closely related to the fact, that update propagation has not been considered. In addition, it is also not possible to define views without any columns, i.e. there exist no Boolean predicates. This concept would be useful for defining general conditional expressions over the database state, i.e., yes/no queries, which would also facilitate the definition of more declarative constraints when applied in conditional clauses of assertions for example:

```
CREATE CONDITION cyclic                CREATE CONDITION two_connections
FROM route r1, route r2                FROM connection c1,c2
WHERE r1.X=r2.Y AND r1.Y=r2.X;         WHERE c1.Y=c2.X AND c1.X<>c2.Y;

CREATE ASSERTION proper_routes
CHECK NOT(cyclic) AND two_connections;
```

In the example, we extend SQL by means of a create condition command used for defining the conditions *cyclic* and *two_connections* introduced above. Assertion *proper_routes* represents the same integrity constraint as the one introduced by the two normative rules in Datalog in the previous section. Note that the check condition becomes quite simple using the predefined conditions. Although SQL can be easily extended by means of Boolean predicates it has to be noted that the approved perception of views as some kind of virtual tables in SQL so far would not hold anymore.

In SQL, column constraints, table constraints and domain constraints must not be defined on views. The definition of domain constraints would not be meaningful since all domains in views are derived from domains of tables. It is, however, possible to have references to views within assertions or check constraints, but you are not allowed to include a check clause into a view definition. Thus, views and integrity constraints cannot be freely combined in SQL making these concepts not orthogonal to each other (see also [BMP01]). The declarative specification of derived concepts by means of deductive rules, however, would directly allow simple specifications

of complex integrity constraints which by now may be expressed by assertions only. In addition it is also not possible to define triggers on views which will be discussed in more detail in the following section. The reason for these restrictions is again a lack of update propagation mechanisms in SQL.

Under certain conditions a view is considered to be updatable; that is, you are allowed to formulate update statements on a view leading to a corresponding update of the underlying base tables. If a view is updatable, updates can still fail due to integrity violations caused by the induced updates. View updates in SQL are restricted to a very small class of views such that updates can be easily translated into base table updates. However, a more general approach allowing arbitrary views to be updated would provide a powerful tool for model generation needed, for example, in expert systems as well as in systems which support advanced integrity repair methods. Apart from the need of efficient top-down and bottom-up propagation mechanisms, a possible reason for restricting view updates in SQL lies in the generation of infinite solutions as the problem of satisfying a given view update request is generally undecidable. On the other hand, infinite cycles are also possible when triggers and user defined functions are taken into account and mechanisms for handling such situations are needed anyway.

2.3 Triggers

Triggers represent a new concept in standard SQL:1999 despite of the fact that active rules had been already implemented in various commercial systems (e.g., DB/2, Oracle and Informix) long before. Triggers allow the specification of a set of SQL statements which are to be executed either before or after tuples of a base table have been updated. The event governing the execution of a trigger is an insert, delete or modification statement applied to a base table, only. Before-triggers are executed before their events and are useful for conditioning of the input data before updates are applied to the database. After-triggers execute after their events and are typically used to implement the reactive behaviour by posing a further insert, delete or modification statement after the updates have been applied. In addition, each trigger has a granularity that defines how many times the trigger is executed for the event. The granularity of a trigger can be specified as either `FOR EACH ROW` or `FOR EACH STATEMENT`, referred to as row-level and statement-level triggers, respectively. The semantics of triggers in SQL was chosen in particular for reconciling the execution of triggers with the evaluation of SQL92 constraints. We will now briefly review the trigger execution model in SQL:1999 with respect to transaction processing and integrity checking in SQL. Afterwards we will discuss oddities and open problems of a possible extension of the trigger concept in SQL.

A transaction in SQL is a sequence of insertion, deletion and modification statements each of them leading to a single update or, in the case of a conditional statement, to a set of updates of one base table. Given an SQL statement M_1 , the corresponding set of updates Δ_1^{up} is calculated with respect to the actual database state. Afterwards, all matching before-triggers are determined; that is, one possible statement-level trigger and a set of row-level triggers. These triggers are executed in the order of their creation time leading to a new set of updates Δ_1^{nup} . These updates are now applied to the database and all relevant immediate integrity constraints are checked. If all immediate integrity constraints are satisfied, a new SQL statement M_2 is determined according to cascading actions of referential integrity constraints leading to a new set of updates Δ_2^{up} . Again a corresponding set of activated before-triggers is determined and executed leading to a new set of updates Δ_2^{nup} . This goes on until the set of updates resulting from referential actions is empty and no integrity violation has occurred. After that, the set of all matching after-triggers is executed including all statement triggers with regard to the considered SQL statements M_1, \dots, M_n and all row-level triggers regarding the entire set of applied updates $\Delta_1^{nup} \cup \dots \cup \Delta_n^{nup}$. The after-triggers are then executed again in the order of their creation time.

When looking at the trigger execution model in SQL:1999 it seems clear that choosing creation time for disambiguating trigger execution is only a temporary solution. First, it is hardly possible to properly control the reactive behaviour of a system when triggers are executed according to that condition. Second, in case there is more than one referential action defined on a modified base table, SQL proposes a nondeterministic approach of evaluating cascading actions leading to an almost arbitrary result if triggers on referenced tables are executed in the proposed order. A possible solution to this problem would be the explicit declaration of priorities within the set of triggers including priorities between triggers defined on different tables. In addition, referential constraints remain problematic as they do not impose a special order of possible cascading actions due to their declarative nature. Therefore it seems more appropriate to define their semantics by means of triggers which themselves may be ordered according to priorities specified before.

If we want to extend the event specification of SQL triggers such that induced updates are allowed as well, the trigger execution model of SQL has to be slightly changed only. An SQL transaction consists of a sequence of SQL statements, each of them leading to a set of updates of one base table only. After each SQL statement within an SQL transaction immediate integrity constraints are checked and activated triggers are processed leading to new SQL statements for updating other base tables. However, already one base table update may lead to a set of induced updates in different views each of them activating their own set of corresponding triggers if triggers on views are allowed. In principal, these triggers are activated at the same time as the ones activated because of the initial modified base table. Thus, the set of before- and after-triggers to be considered ought to include all activated triggers of the modified base table as well as all triggers activated by the corresponding induced updates. These sets of triggers have to be ordered for processing according to a given criterion. This again could be the explicit specification of priorities between triggers or their creation time. Integrity checking then would be necessary for the modified base table as well as for the 'modified' views if constraints on views were allowed. This model for integrating the execution of triggers with the evaluation of constraints in SQL can be specified by means of a fixpoint evaluation. A fixpoint is reached when all integrity constraints are satisfied and all triggers are executed.

3 Conclusion

This paper proposes extensions of SQL making the concepts views, triggers and integrity constraints really orthogonal to each other. The extensions discussed do not enhance the expressivity of SQL, but may be extremely useful for designing more advanced database applications. Recent results in the field of deductive databases have shown that a fixpoint based inference mechanism could be a possible basis for implementing this new functionality. Nevertheless, many open problems remain as it is not easy to find a sound extension of SQL, especially when transferring results from a different context such as deductive or active databases.

References

- [BMP01] BEHREND, A., MANTHEY R., PIEPER B.: *An Amateur's Introduction to Constraints and Integrity Checking in SQL3*. BTW 2001: 405-423.
- [CW90] CERI S., WIDOM J.: *Deriving Production Rules for Constraint Maintainance*. VLDB 1990: 566-577.
- [Gri97] GRIEFAHN, U.: *Reactive Model Computation - A Uniform Approach to the Implementation of Deductive Databases*. Dissertation, University of Bonn, 1997.
- [Man94] MANTHEY, R.: *Active and passive rules in database systems: How do they relate?* ADBIS 1994: 104-115, Moscow (Russia), June 1994 .
- [Pie01] PIEPER, B.: *Inkrementelle Integritätsprüfung und Sichtenaktualisierung in SQL*. Dissertation, University of Bonn, 2001.