In: *IEEE Data Engineering Bulletin, Special Issue on Active Databases*, 15(4):15–18, Dec. 1992

The Starburst Rule System: Language Design, Implementation, and Applications

Jennifer Widom

IBM Almaden Research Center *

Abstract

This short paper provides an overview of the Starburst Rule System, a production rules facility integrated into the Starburst extensible database system. The rule language is based on arbitrary database state transitions rather than tuple- or statement-level changes, yielding a clear and flexible execution semantics. The rule system was implemented rapidly using the extensibility features of Starburst; it is integrated into all aspects of query and transaction processing, including concurrency control, authorization, recovery, etc. Using the Starburst Rule System, we have developed a number of methods for automatically generating database rule applications, including integrity constraints, materialized views, deductive rules, and semantic heterogeneity.

1 Introduction

The *Starburst Rule System* is a facility for creating and executing *database production rules*; it is fully integrated into the Starburst extensible relational database system at the IBM Almaden Research Center. Production rules in database systems (also known as *active database systems*) allow specification of database operations that are executed automatically whenever certain events occur or conditions are met. In most active database systems, including Starburst, production rules are a persistent part of the database and are created using a *rule definition language*. As users and applications interact with data in the database, rules are triggered, evaluated, and executed automatically by a *database rule processor*. In developing the Starburst Rule System we had two major goals:

- Design of a rule definition language with a clearly defined and flexible execution semantics
- Rapid implementation of a fully integrated rule processor using the extensibility features of Starburst

As we developed and experimented with our language and system, we discovered that the inherently unstructured nature of rule processing makes production rules quite difficult to program. Consequently, we added as a third goal:

- Development of methods for specifying common classes of database rule applications in high-level languages and compiling these specifications into Starburst rules

The remaining three sections of this short paper outline the approaches we have taken to meeting each of these three goals. Further details on language design appear in [WF90, Wid92], further details on implementation appear in [WCL91], and further details on applications appear in [CW90, CW91, CW92a, CW92b, Wid91].

2 Language Design

There are two important aspects in the design of a database production rule language: the syntax for creating (as well as modifying, deleting, and grouping) rules, and the semantics of rule processing at run time. Most database production rule languages have a similar syntax, relying on and extending the syntax of the database query language. However, the semantics of rule processing varies considerably.

In Starburst, the syntax for creating a rule is:

> **create rule** *name* **on** *table*
> **when** *triggering operations*
> [**if** *condition*]
> **then** *action*
> [**precedes** *rule-list*] [**follows** *rule-list*]

*Address: 650 Harry Road, San Jose, CA 95120 E-mail: widom@almaden.ibm.com

The *triggering operations* are one or more of **inserted**, **deleted**, and **updated**$(c_1, .., c_n)$, where $c_1, .., c_n$ are columns of the rule's *table*. The optional *condition* is an arbitrary SQL predicate over the database. The *action* is an arbitrary sequence of database operations, including SQL data manipulation commands, data definition commands, and **rollback**. The optional **precedes** and **follows** clauses are used to partially order the set of rules: if a rule r_1 specifies a rule r_2 in its **precedes** list, or if r_2 specifies r_1 in its **follows** list, then r_1 has higher priority than r_2. Commands also are provided to **alter**, **drop**, **deactivate**, and **activate** rules. *Rule sets* may be created; each set contains zero or more rules, and each rule belongs to zero or more sets.

Rules are processed at *rule processing points*. There is an automatic rule processing point at the end of each transaction, and there may be additional user-specified processing points within transactions. We first describe end-of-transaction rule processing. The semantics is based on *transitions*—arbitrary database state changes resulting from execution of a sequence of SQL data manipulation operations. The state change created by the user transaction is the first relevant transition, and some rules are triggered by this transition. As triggered rule actions are executed, additional transitions are created which may trigger additional rules or trigger the same rules additional times. Rule processing is an iterative algorithm in which:

1. A triggered rule R is selected for *consideration* such that no other triggered rule has priority over R (for further details on Starburst's rule ordering strategy see [ACL91])

2. R's condition is evaluated

3. If R's condition is true, R's action is executed

For step 1, a rule is triggered if one or more of its triggering operations occurred in the composite transition since the last time the rule was considered, or since the start of the transaction if the rule has not yet been considered. The effect of this semantics is that each rule sees each modification exactly once. Rule processing terminates when a **rollback** action is executed (in which case the entire transaction aborts), or when there are no more triggered rules.

Within a transaction, rule processing can be initiated by commands **process rules**, **process ruleset** S, or **process rule** R. Command **process rules** invokes rule processing with all rules eligible to be considered and executed; command **process ruleset** S invokes rule processing with only rules in set S eligible to be considered and executed; command **process rule** R invokes rule processing with only rule R eligible to be considered and executed. The semantics of rule processing in response to each of these commands is identical to end-of-transaction rule processing.

Rule conditions and actions may refer to the current state of the database through top-level or nested SQL **select** operations. In addition, rule conditions and actions may refer to *transition tables*, which are logical tables reflecting the changes that have occurred during a rule's triggering transition. Transition table **inserted** in a rule refers to those tuples of the rule's table that were inserted by the triggering transition; transition tables **deleted**, **new-updated**, and **old-updated** are similar.

3 Implementation

The Starburst rule language as described in Section 2 is fully implemented, with all aspects of rule definition and execution integrated into normal database processing. The implementation took about one woman-year to complete; it consists of about 28,000 lines of C and C++ code including comments and blank lines (about 10,000 semicolons). The implementation relies heavily on several extensibility features of the Starburst database system [H+90].

We briefly outline the rule system's general design; many details necessarily are omitted. Rule and rule set information is stored in *rule catalogs*, portions of which are cached in global main memory structures (i.e. structures shared by all processes). During query processing, the system monitors data modifications that may trigger rules. Monitoring takes place using Starburst's *attachment* extensibility feature: based on the current set of rules, attachment procedures are registered by the rule system to be called on relevant tuple-level insert, delete, and update operations. These procedures enter the modifications in a local main memory structure (i.e. one structure per process) called a *transition log*. Automatic end-of-transaction rule processing is invoked by Starburst's *event queue* extensibility feature: if a transaction may trigger rules, a rule processing procedure is placed on an event queue to be invoked at the commit point of the transaction. (Hence, there is no overhead at all if a transaction does not trigger rules.) Rule processing also may be invoked by user commands, as described in Section 2. During rule processing, triggered rules are determined using the transition log, and they are stored in a local main memory sort structure reflecting their ordering. The Starburst query processor is called to evaluate rule conditions and execute rule actions. Transition tables are implemented using Starburst's *table function* extensibility feature: table functions are referenced as tables in SQL but their contents are generated at run time by registered procedures. The rule system registers four procedures (*inserted*, *deleted*, *new-updated*, and *old-updated*) that produce the transition tables from the transition log as needed during condition evaluation and action execution. Finally, note that since the transition log is central to rule processing, it is highly structured for efficiency in its various operations.

A few special features were needed to fully integrate the rule system into Starburst. Since the query processor is called to execute rule conditions and actions, concurrency control for these operations is handled automatically. However, concurrency control for rule creation, modification, and deletion must be handled separately. The rule system includes concurrency control mechanisms that ensure consistency with respect to rules and data (i.e. the set of rules triggered by a given transaction's modifications cannot change during the course of the transaction) and with respect to rule ordering (i.e. the ordering between triggered rules cannot change during the course of rule processing). The rule system also includes an authorization component for rules and rule sets, and rollback recovery mechanisms for rule system data structures.

4 Applications

The Starburst Rule System provides a powerful mechanism that can be used for traditional database functions such as integrity constraints and derived data, for non-traditional database functions such as situation monitoring and alerting, and as a platform for large knowledge-base and expert systems. Unfortunately, developing a set of rules to correctly realize such applications can be a difficult task: rule processing is inherently dynamic and unstructured, it interacts with arbitrary database changes, and its behavior can be unpredictable and difficult to specify.

We have taken two approaches to the problem of developing rule applications. In the first approach, support is provided to the rule programmer in the form of *analysis tools*. These tools perform static analysis on a set of Starburst rules to predict (conservatively) whether the rules are guaranteed to terminate, whether they are guaranteed to produce a unique final database state independent of the ordering between non-prioritized rules, and whether they are guaranteed to produce a unique stream of "observable" actions independent of the ordering between non-prioritized rules; this work is reported in [AWH92]. Our second approach is based on the observation that, unlike rules themselves, many common rule applications are static, structured, and easy to specify. Hence, we have developed a suite of methods whereby the user can specify rule applications using a high-level declarative language, and these specifications are translated (fully- or semi-automatically) into rules that implement them. We briefly describe four such classes of applications.

Integrity constraints – Integrity constraints are static predicates over the database that must be true at certain *consistency points* (usually the end of each transaction). Starburst rules can be used to monitor and enforce integrity constraints: for each constraint, a rule is triggered by any database modifications that may violate the constraint, the rule's condition checks whether the constraint actually is violated and, if the condition is true, the rule's action restores the constraint or rolls back the transaction. We have developed a method whereby the user specifies constraints as SQL predicates over the database. From an arbitrary set of constraints, the system semi-automatically derives a set of rules that are guaranteed to maintain the constraints [CW90].

Materialized views – Views are logical tables specified as queries over *base* (stored) tables. When a view is materialized, the view table is stored in the database and must be kept consistent with the base tables. Starburst rules can be used to maintain materialized views: whenever base table modifications may affect the value of a view, rules are triggered whose actions modify the view accordingly. We have developed a method whereby the user specifies views using SQL, then the system automatically derives a set of Starburst rules that are guaranteed to maintain materializations of the views in an incremental fashion [CW91].

Deductive rules – Similar to views, deductive rules specify logical tables derived from base tables. However, deductive rules use a recursive logic programming formalism, which is more powerful than SQL views. Similar to materialized views, Starburst rules can be used to maintain materialized derived tables specified by deductive rules: whenever base table modifications may affect the value of a derived table, rules are triggered whose actions modify the derived table accordingly. If derived tables are non-materialized (i.e. produced on demand rather than stored in the database), then Starburst rules can be used to perform the iterative *semi-naive* evaluation mechanism often used for deductive rules. We have developed methods for automatically deriving Starburst rules from deductive rules for both of these approaches [CW92a, Wid91].

Semantic heterogeneity – Semantic heterogeneity occurs when multiple databases model the same real-world entities in different ways. Whenever possible, it is desirable to maintain consistency across such databases, despite the heterogeneity. While Starburst rules alone are not sufficient for this, they can be used together with a *persistent queue* mechanism. At each database, rules are triggered by any modifications that may violate consistency with other databases. Semantic heterogeneity is encoded in rule conditions and actions so they can perform remote read operations to determine whether consistency actually is violated and, if so, perform local or remote update operations to restore consistency. Since multidatabase environments usually do not support transactions across sites, persistent queues are used for reliable and correct execution of remote operations. We have developed a method whereby the user specifies consistency requirements across semantically heterogeneous databases in a high-level language, then the system automatically derives Starburst rules that are guaranteed to monitor and enforce consistency [CW92b].

Acknowledgements

I am ever grateful to Stefano Ceri, Bobbie Cochrane, Shel Finkelstein, and Bruce Lindsay, all of whom made important contributions to one aspect or another of the Starburst Rule System.

References

[ACL91] R. Agrawal, R.J. Cochrane, and B. Lindsay. On maintaining priorities in a production rule system. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 479–487, Barcelona, Spain, September 1991.

[AWH92] A. Aiken, J. Widom, and J.M. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 59–68, San Diego, California, June 1992.

[CW90] S. Ceri and J. Widom. Deriving production rules for constraint maintenance. In *Proceedings of the Sixteenth International Conference on Very Large Data Bases*, pages 566–577, Brisbane, Australia, August 1990.

[CW91] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 577–589, Barcelona, Spain, September 1991.

[CW92a] S. Ceri and J. Widom. Deriving incremental production rules for deductive data. IBM Research Report, IBM Almaden Research Center, November 1992.

[CW92b] S. Ceri and J. Widom. Managing semantic heterogeneity with production rules and persistent queues. IBM Research Report, IBM Almaden Research Center, October 1992.

[H+90] L. Haas et al. Starburst mid-flight: As the dust clears. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):143–160, March 1990.

[WCL91] J. Widom, R.J. Cochrane, and B.G. Lindsay. Implementing set-oriented production rules as an extension to Starburst. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 275–285, Barcelona, Spain, September 1991.

[WF90] J. Widom and S.J. Finkelstein. Set-oriented production rules in relational database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 259–270, Atlantic City, New Jersey, May 1990.

[Wid91] J. Widom. Deduction in the Starburst production rule system. IBM Research Report RJ 8135, IBM Almaden Research Center, San Jose, California, May 1991.

[Wid92] J. Widom. A denotational semantics for the Starburst production rule language. *SIGMOD Record*, 21(3):4–9, September 1992.