Practical Applications of Triggers and Constraints: Successes and Lingering Issues

Stefano Ceri*
Politecnico di Milano
ceri@ipmel2.elet.polimi.it

Roberta J. Cochrane
IBM Almaden Research Center
bobbiec@almaden.ibm.com

Jennifer Widom†
Stanford University
widom@cs.stanford.edu

1 Introduction

From about the mid-1980's to the mid-1990's there was a flurry of research activity in the area of database triggers and constraints, seeing the development of numerous research proposals and prototypes. Soon thereafter, most mainstream database products ramped up their support for constraints and triggers, with expressive constraint specifications appearing in the SQL-92 standard, and both constraints and triggers in the SQL-99 standard.

We briefly review the emergence of research in constraints and triggers, and we briefly describe standards and current commercial support. We then focus on practical applications of triggers. We describe a variety of interesting and significant ways in which triggers have been put into practice, and we classify trigger applications along two dimensions: *handcrafted* versus *generated*, and *kernel DBMS* versus *DBMS services* versus *external applications*. We also argue that a significant portion of these trigger applications are in fact nothing more than *constraint-maintainers* for various classes of integrity constraints, indicating that our work a decade ago [CW90]—if not itself put into practice directly—was not far off the mark. Finally, we analyze the evolution of trigger applications and discuss some lingering shortcomings in database constraint and trigger systems.

2 Brief Research History

We begin with a very brief description of the emergence and development of constraints and triggers as research topics within the database field.

*Supported in part by the EC under grant P28771 W3I3
†Supported in part by the NSF under grant IIS-9811947

**Proceedings of the 26th VLDB Conference,
Cairo, Egypt, 2000.**

2.1 Constraints

The idea of *integrity constraints* in relational databases appeared not long after the relational model itself, with several foundational papers in 1975 (the year of the first VLDB conference, incidentally) [EC75, HM75, Sto75]. After this initial work within the research community, database products have steadily provided increasing support for constraints, as discussed in Section 3.1 below.

Integrity constraints also have provided tremendous fodder for database research—we will certainly not attempt to provide a survey here. Suffice it to say that because constraints are theoretically well-grounded (as Boolean predicates), and at the same time are of great practical significance (as preservers of database integrity), researchers from all corners of the database field have made contributions ranging from deep theorems to significant systems.

2.2 Triggers

Shortly after researchers recognized the importance of database integrity constraints, including automatic "reactions" to constraint violations, the idea expanded to the more general concept of *triggers* [Esw76], also now known as *event-condition-action* (*ECA*) *rules* or *active rules*. However, triggers as a research field—or as a feature of commercial database systems for that matter—did not take off nearly as quickly as constraints. One can speculate as to the underlying reasons for the delay; our hunch is that triggers were not as in demand by database users, and as a research topic triggers are not nearly as well-defined or easily grounded as integrity constraints.

It was not until the mid-to-late 1980's that the area of triggers, by then referred to as *active database systems*, truly came alive, and it did so with gusto. A number of significant research efforts were launched, and in the early 1990's there was little doubt that active databases were considered one of the "hot topics" in database research. Interest remained high for several years: products launched simple trigger systems while researchers prototyped more elaborate and expressive ones, and some theoretical work emerged as well. Again, we will not attempt to provide a survey, but we refer the reader to [WC96] for a snapshot of the field in the mid-1990's.

3 Standards and Products

Let us now briefly examine how constraints and triggers have developed in commercial database products, and discuss their standardization. We will go into a bit more depth than in Section 2 since the core topic of this paper is applications built within or upon commercial support, but again, we are not attempting to provide a comprehensive survey or tutorial.

3.1 Constraints

The SQL-92 (and subsequent SQL-99) standard provides several mechanisms for specifying integrity constraints. The most common kinds of constraints—*keys*, *non-null constraints*, and *referential integrity*—each have their own syntax and enforcement mechanisms [UW97]. Of interest in relationship to triggers is the fact that referential integrity constraints can be specified with particular actions to be taken upon violations, such as *cascaded delete* or *set null*.

The more general *check* constraints are associated with a given database table. SQL-like syntax is used to specify conditions that must hold for each tuple of the table, and the conditions are checked on inserts and updates to the table. Although the SQL-92 standard permits subqueries within *check* constraints, thereby enabling *check* constraints to be used for multi-tuple and multi-table constraints, most products do not support this feature. General constraints that are not specific to a single table can be specified by SQL-92 *assertions*, although again many products do not support this level of generality. Finally, *domain* constraints can be specified to constrain all values in all columns of the domain, or any value cast to the domain.

Although constraints are specified declaratively, every constraint implicitly specifies:

- A set of *events* after which the constraint is checked—generally any database operation that could cause the constraint to become violated.
- An *action* to be taken if the constraint is violated—usually raising an error and/or generating a rollback, with some more interesting cases such as referential integrity as described above.

When a constraint is first defined or when new data is loaded, the system verifies the constraint against the data. Thereafter the *events* are monitored and *actions* executed to ensure that the database state always satisfies the constraint as specified.

3.2 Triggers

Although trigger support was not included as part of the SQL-92 standard, triggers were supported by some products already in the early to mid-1990's. The SQL-99 standard has extensive coverage of triggers, and today all major relational DBMS vendors have some support for triggers. Unfortunately, because the standard was influenced by preexisting product support, and many products do not do a good job integrating constraints and triggers, most products support only a subset of the SQL-99 trigger standard and most do not adhere to some of the more subtle details of the execution model [CPM96]. Furthermore, some trigger implementations rely on proprietary programming languages for specifying parts of their triggers, which makes portability across different DBMS's difficult.[1]

In contrast to declarative constraints, triggers are explicitly procedural. A trigger is activated whenever a specified *event* occurs, usually an *insert*, *delete*, or *update* on a particular table. Once activated, an optional specified *condition* is checked and, if the condition is true (or omitted), an *action* is executed. There are a number of important details to the specification and execution semantics of triggers, only a few of which are covered here.

Triggers have an *activation time* (either *before*, *after*, or *instead of* the triggering event), and a *granularity* (either *row-level* or *statement-level*). There are some obvious as well as some subtle distinctions in the way triggers behave depending on which settings are selected. Each trigger has access to the old and new values of the row or statement affected by the event, by means of *transition variables* (`OLD` and `NEW`) and *transition tables* (`OLD_TABLE` and `NEW_TABLE`). Conditions can be arbitrary predicates, and actions are stored procedures which may include SQL statements, control constructs, and calls to user-defined functions. Note that user-defined functions invoked by a trigger action may have side effects that fall outside of DBMS control (including possibly calling the DBMS itself).

Trigger support in DBMS products is variable, with typical deviations from the standard including, e.g., restrictions on predicates in trigger conditions, restrictions on references to transition variables or tables, raising exceptions after a certain number of trigger activations, and trigger actions specified using proprietary languages as discussed above.

3.3 Constraints and Triggers in Products

Products also vary considerably in terms of their integration of constraint and trigger facilities. Historically, some products (e.g., Sybase) offered only triggers, relying on applications to implement any declarative constraints they needed using triggers (perhaps following the methodology of [CW90]). Eventually built-in constraints were added to these products for performance, usability, and to conform to the SQL standard. At the same time, other products (e.g., IBM DB2) initially supported constraints only. Although very expressive declarative constraints were allowed, trig-

[1] We expect this last issue to dissolve if the *Persistent Stored Modules* (*SQL-PSM*) language becomes standardized, or if a language such as Java becomes adopted widely for database procedures.

	Embedded in DBMS Kernel	DBMS Services	External Applications
Handcrafted	Metadata management, Internal audit trails	N/A	Business rules, Scheduling, Supply chain management Web applications
Generated	Referential integrity, Materialized views	Replication, Extenders, Audit trails, Migration, Alerters	Workflow management

Figure 1: Classification and examples of trigger applications

gers were added to these products eventually based on application needs, and again to conform to the standard.

The important point to note is that the marketplace has dictated that separate support for both constraints and triggers is appropriate.

4 Trigger Applications

Now that we have reviewed triggers and constraints briefly, both from a research and commercial standpoint, let us take a look at how the functionality has been deployed in real applications.

In Figure 1 we characterize trigger applications along two dimensions. The vertical dimension distinguishes between those triggers that are written by hand for a specific application (*handcrafted*), versus "generic" trigger sets that are produced automatically for a specific purpose, usually parameterized for a given application (*generated*). In the horizontal dimension, on the far right are applications that reside entirely outside of the DBMS, creating triggers and (possibly) responding to trigger actions through the database system's client API. The middle column represents trigger applications that are constructed by the DBMS vendor or a third-party, generally to provide a service or to enhance a specific database functionality. The far left column represents trigger-like behavior built into the kernel of the DBMS. For the last class of applications, the trigger system of the DBMS may be used to prototype the desired functionality, but eventually the behavior is hard-coded into the kernel, in order to circumvent security restrictions, achieve higher performance, or to program declarative behavior that procedural triggers cannot simulate with complete accuracy.

Although not perfect, the dimensions in Figure 1 enable us to classify trigger applications as well as to characterize their evolution. In Sections 5 and 6 we discuss generated and handcrafted trigger applications, respectively. Then in Section 7 we discuss the general evolution of trigger applications, and in Section 8 we attempt to provide a more fine-grained classification than in Figure 1.

5 Generated Triggers

Our own early work established that triggers can be generated automatically for a wide class of applications, including constraint maintenance [CW90], materialized view maintenance [CW91], and managing semantic heterogeneity [CW93]. An entire database design framework based on trigger generation is presented in [CF97]. The primary idea behind all of this work is that in many cases the desired end result of trigger behavior can be specified declaratively (e.g., *maintain this constraint*, or *keep this view consistent with the base data*), and a set of procedural triggers can be generated automatically from the declarative specifications. This approach can guarantee correctness (which is no minor matter when it comes to triggers), and it frees the user from the detailed and error-prone task of constructing a trigger set by hand. We will argue that a large fraction of useful trigger applications can be approached in this manner—in fact many of them fall into the more specific constraint-maintaining category—even if such triggers are hard-coded today.

5.1 Internal Generated Triggers

As pointed out by Figure 1, two classic instances where triggers can be used to support kernel database functionality are *referential integrity* and *materialized views*. As discussed in Section 3.1, referential integrity is the only built-in constraint type that allows a variety of different actions to occur when the constraint is violated, with the desired actions specified declaratively by the user. Generating a set of triggers to support a single referential integrity constraint with any of the available actions is a straightforward exercise (one that we often assign in our introductory database courses), although there are some subtleties to maintaining multiple referential integrity constraints using triggers [Hor92]. It also is possible to generate a set of triggers that will keep a materialized view consistent with the base data—either naive triggers that recompute the view, or more complicated ones that maintain it incrementally [BDD+98, CW91, LSPC00].

For both referential integrity and materialized views, triggers are a natural and easy mechanism for implementing the desired functionality: with a trigger system in hand, one can provide referential integrity and materialized view support in no time. However, these features also are very intrinsic to database performance, and they are tied up with authorization and transactional issues as well. As a result, most DBMS's will select to implement separate, special-purpose, highly-tuned components for referential integrity and materialized views. It would certainly be nice if trigger systems were fast, scalable, and flexible enough to be used for kernel activities instead of hard-coding them, with no loss of performance or functionality. In the meantime, triggers still provide an excellent means of rapidly prototyping functionality that may end up hard-coded within the kernel of the DBMS.

5.2 Generated Triggers for Services

Moving away from the database kernel, we come to one of the widest and most useful classes of trigger applications: those that can be generated automatically to support a feature or service that enhances the functionality of a database system (second row, middle column of Figure 1). In fact, some purveyors of early "extensible" database systems, which led to today's prevalent object-relational DBMSs, suggested that two of the main features comprising extensibility were objects and active rules [LLPS91, SK91, SRL$^+$90]. Trigger-based services in this class can be designed and implemented by the database system vendor, or provided by third-parties.

Three example applications in this class noted in Figure 1 but not discussed beyond this paragraph are *audit trails*, *migration*, and *alerters*. It should be clear that automatically-generated triggers can easily be used to maintain logs that capture database activity for auditing purposes. In fact, audit trails were one of the earliest suggested applications of triggers, and the ability to generate audit trails still remains a "benchmark" for trigger languages and systems.[2] Triggers also can be used during the process of migrating data from one table or schema (or even DBMS) to another, to ensure consistency when updates occur during the migration process. This application is similar to replication, discussed momentarily in Section 5.2.1. Finally, *alerters* allow users to register for certain database conditions to become true, in which case the user is notified and data may be sent along with the notification, but no action on the database itself is usually taken. Note that in some ways alerters are very similar to integrity constraints, except in the case of alerters a condition becoming true results only in a message being sent, rather than in an error and/or rollback. Furthermore, we expect that alerters may generate orders of magnitude more trigger instances on a given table, as discussed in Section 9 below.

5.2.1 Replication

Most database systems include features for replicating data automatically between tables in a given database schema, across schemas within the same database server, or across servers and even vendors. Tables may be replicated exactly, or the replicated tables may be specified as more complicated expressions (views, essentially) over the source tables [Tho97]. In all cases, the fundamental operation is to capture changes at source tables and propagate them to replicas. This application shares many obvious similarities with materialized views and can similarly rely on automatically-generated triggers, but it requires interactions outside of the kernel DBMS while materialized views are primarily internal.

Triggers are widely used for the "capture" phase of replication services at a minimum. Triggers also may be used for the "propagate" phase, although because propagation may be decoupled from the transactional semantics of the underlying database systems, in some cases triggers may not be expressive or flexible enough to achieve the desired behavior. For example, in many systems the propagation of updates to replicas is purely time-driven, and most trigger systems currently do not support time-based events.

5.2.2 Extenders

Another widely deployed service supported by triggers is maintaining data structures stored either internally or externally to the database (e.g., specialized indexes) that need to stay consistent with base data stored in the database. Triggers capture changes to the base data and propagate them to the specialized structures. More generally, triggers can provide the "glue" for applying any specialized functions, both internal and external to the DBMS, to specialized data stored in the DBMS. There are numerous *extenders* of this form relying on triggers today, for example we immediately counted eleven that we know of developed by outside vendors to run on IBM's DB2. A few examples are discussed briefly in the next paragraph.

Extenders for multimedia data (image, audio, video, etc.) all use automatically-generated triggers. Triggers are used on insertions to validate multimedia input and generate useful metadata. Other triggers are used subsequently to keep data and metadata synchronized. Triggers are also used heavily in text extenders: When a row is inserted containing a text attribute, a trigger will automatically create a handle for the text, place the text in a specialized external index, and place the handle in the actual base row. Other triggers will ensure that the level of indirection is followed, and will keep the external text index consistent when text data is modified. XML extenders behave similarly, although the trigger actions are more complex. XML documents can be parsed and validated automatically by triggers; document components are then separated and stored in specialized indexes to enable structural searches.

5.3 External Generated Triggers

Generating triggers from declarative specifications is a natural approach to take for DBMS services as discussed in the previous section, since such services tend to be parameterized, only moderately configurable, and relatively simple to specify. For trigger applications that are completely

[2] The main issue here is that if one is interested in a complete audit trail (including actions that may ultimately be rolled back), triggers must have the ability to be activated by uncommitted events.

external to the DBMS, automatic trigger generation is less prevalent, although one important example in this class is *workflow management*.[3]

It is interesting to note that workflow management was one of the earliest suggested applications of expressive triggers [DHL90], although initial work did not propose automatic trigger generation. In recent developments in the commercial sector, Oracle's *Workflow Builder* [Ora00] and Informix's *Media360* package [Inf00] both provide tools that generate triggers automatically from higher-level workflow specifications, an approach also suggested in [BBC+97].

6 Handcrafted Triggers

Handcrafted triggers generally support logic that is very specific to the application at hand. In many cases, handcrafted trigger applications cannot be expressed declaratively. In some cases, even if a declarative specification is possible, it simply may not be worthwhile to write a "trigger generator" if the trigger set will be instantiated only a small number of times. We have found that most handcrafted trigger applications reside entirely outside of the DBMS, i.e., the upper-right entry in Figure 1, and we were unable to uncover any handcrafted trigger applications providing a DBMS service (the upper-middle entry)—by nature, services tend to be parameterized and simple enough that they can be specified declaratively.

6.1 Internal Handcrafted Triggers

Two uses for handcrafted triggers within the kernel of a DBMS are for *metadata management*, and to maintain customized *internal audit trails* for system administrators. As an example of using internal triggers for metadata management, we are familiar with the details of system catalogs in IBM's DB2 product. For performance, DB2 maintains complex, optimized internal data structures (*descriptors*) that are used during query compilation. These descriptors are derived from the values of other catalog attributes, and they encode (among other things) table statistics. When table statistics are updated, triggers propagate the new statistical values to keep the descriptors consistent. While the function of these triggers is to maintain integrity constraints among catalog data, the details of the data structures are complex, low-level, and subject to adjustments, so it made sense to handcraft a set of triggers to propagate updates in an efficient and correct manner.

6.2 External Handcrafted Triggers

Handcrafted, external triggers are in some sense the most straightforward deployment of triggers, although these applications also can be the most error-prone. Each trigger or set of triggers is written by an application developer to support application-specific logic that can be managed in part by the DBMS. Often, such triggers are used simply to invoke functions that perform actions external to the DBMS. However, external handcrafted triggers also may be used to maintain auxiliary information within the database that is dependent on the application and may not be easy to specify declaratively. For example, triggers might compute derived columns for each tuple in a table using application-defined "black box" computations, or they might tag inserted and updated rows with a timestamp and/or operation type.

There are a number of broad classes of external handcrafted trigger applications worth noting, as shown in Figure 1: *business rules*, *scheduling*, *supply chain management*, and *Web applications*. In each of these classes, although the overall goal may be the same across applications, the specifics of a given application—e.g., the business logic of a particular enterprise, the constraints of a particular scheduling problem—vary enough that a generic system for generating triggers from a declarative specification does not seem feasible.

The fact that handcrafting triggers is difficult has been recognized for quite some time with no broadly applicable solution to date. There is clearly room for trigger programming "wizards" that:

1. Allow the application developer to specify an external trigger-based application in a higher-level language.
2. Translate the specification into triggers.
3. Provide some analysis tools for identifying how the generated triggers will interact.

This approach follows the general approach of [CW90, CW91, CW93], but we are suggesting its use for applications that are somewhat less declarative than the applications discussed in those papers, i.e., the tools would target applications that are nearly always handcrafted today. Some headway has been made in this direction for business rules [Ros97], but primarily applying to business rules that effectively enforce integrity constraints.

We attempted to quantify the number of triggers that are typical in a handcrafted external application, in part to determine whether weak points in trigger scalability are relevant. In one scheduling application that we studied, there were a total of 25 triggers defined over 6 tables. About half of the triggers were defined over two auxiliary tables that were needed to encode the application logic and control the firing of other triggers. Even with this relatively modest number of triggers, they were difficult to write and cumbersome to maintain, and inadvertently contained avoidable recursive logic. By contrast, in a Web application we

[3] Admittedly, *business rules* and *scheduling* are similar applications that might use automatic trigger generation, but to date these applications have not done so to the extent of workflow, so we leave business rules and scheduling in the "handcrafted" category.

studied there were over 100 triggers. However, the triggers were spread across nearly 30 tables and served largely to log relevant updates, so trigger interactions were not a serious problem.

7 Evolution of Trigger Applications

Early trigger applications tended to be external and handcrafted. Trigger systems were touted as the mechanism that would allow databases to become "knowledge bases," with built-in rule-based reasoning capabilities. This claim turned out to be too strong for at least two reasons: trigger performance was not adequate for the large number of rules anticipated in deployed knowledge bases, and trigger interactions were cumbersome and difficult to debug or formally verify.

Our work in the early 1990's [CW90, CW91, CW93] showed that declarative specifications could be used to generate triggers for several useful applications. With this approach, correctness is guaranteed, regardless of the number of (generated) triggers in the application. As trigger applications have "settled out", we see the general trend reflected in Figure 1. Outside of the DBMS kernel, many applications fall into one of two categories: either the triggers are generated automatically for a parameterized service provided in conjunction with the database system (e.g., replication), or the triggers are written by hand for a specific application-dependent task (e.g., scheduling).

Anecdotal evidence suggests that even though triggers have been used to great success for a wide variety of applications as discussed above and shown in Figure 1, it is still the case that a preponderance of trigger applications are simply maintaining relatively straightforward integrity constraints. As a result, scalability in number of triggers has not been a significant concern, with performance efforts devoted instead to scalability in the amount of data that (a small set of) triggers operates over. As triggers become more widely deployed for alerting systems, complex scheduling tasks, and other applications requiring numerous and/or complex triggers, scalability is likely to become a significant concern [BBC+97].

8 Further Classification

We now propose a classification of triggers that is based largely on function and behavior, rather than on the dimensions in Figure 1. The first 8 of these categories are usually generated triggers, while the last are typically handcrafted.

1. **Constraint-preserving triggers:** Signal integrity constraint violations and force rollbacks of the violating transactions.

2. **Constraint-restoring triggers:** Detect integrity constraint violations and modify the database contents in order to restore integrity.

3. **Invalidating triggers:** Signal and mark integrity constraint violations, allowing applications to respond appropriately.

4. **Materializing triggers:** Compute materialized derived information, from simple scaler values to aggregate values to complex views, either by incremental modifications or complete refresh.

5. **Metadata triggers:** Maintain consistency across system catalogs or other metadata (recall Section 6.1).

6. **Replication triggers:** Replicate, migrate, or log information and/or modifications from one table or database (the *primary copy*) to another one (the *secondary copy*).

7. **Extenders:** Manage new types of data (e.g., validate input) and keep specialized external structures consistent with the base data.

8. **Alerters:** Notify or push information to users in the form of messages, typically based on a publish/subscribe model.

9. **Ad-hoc triggers:** Implement business rules, scheduling, workflow, supply-chain management, or other application-specific logic.

We can see that types 5, 6, and 7 are clearly derivatives of type 4, and type 4 itself can be thought of as a specific instance of type 2. In other words, for many trigger applications, the primary purpose is to monitor and maintain some kind of constraint. Furthermore, if we stretch our imagination a bit, other trigger types also can be thought of as constraint maintainers. For example, alerting triggers (type 8) are responding to the abstract constraint that each user must be aware of the information for which he subscribed, perhaps within a certain amount of time.

Once a trigger application can be expressed in the context of maintaining constraints, a framework based on [CW90] may be applicable. In such a framework, trigger generation must enumerate the events that can cause the constraint to be violated, then associate a corrective action for each event. Correctness of the approach is guaranteed if all possible sources of inconsistency are covered. Once triggers for each class of constraints are generated, we might integrate the different types of constraint-maintaining triggers into a coherent framework. For example, we might want to first restore integrity constraints on base data, then update materialized views, then restore integrity constraints on views, then perform replication, and finally execute alerters.

9 Lingering Issues

The 1995 paper by Simon and Kotz-Dittrich [SKD95], which was based on the author's practical experience deploying trigger-based applications, did a good job of summarizing both the benefits and pitfalls of database triggers. To a large extent, the positive and negative aspects brought forth in that paper remain true today. The main recognized advantages of trigger-based applications are still the ability to move shared application logic and business rules into the database (rather than hard-coding the behavior into all applications), and the ability to specify integrity constraints that go beyond the specific types of built-in constraints supported by SQL.

On the negative side, among the problems cited by [SKD95] based on their application-building experience are:

1. Lack of expressive events in the SQL standard, e.g., no event predicates, user-defined events, or conjunction.

2. Product limitations, such as a maximum number of triggers per event type.

3. Lack of uniformity across products—syntactic, semantic, and transactional—resulting in confusion and a lack of portability.

4. Subtle behavior, particularly when mixing different types of triggers, or mixing triggers and built-in constraints.

5. Lack of structuring mechanisms and debugging tools for triggers, making it very difficult to specify and understand how a large number of triggers interact among themselves and with transactions.

6. Performance penalty when compared against hard-coding and optimizing the desired effect.

Let us examine how each of these problem areas has evolved since 1995.

Expressive events

Events are no more expressive, and the now-established SQL-99 standard with only the simplest of event types (*insert*, *delete*, or *update* on one table, possibly restricted to certain columns) virtually guarantees that events will remain simple. However, richer event types as suggested in [SKD95] can largely be encoded using SQL-99 triggers by making trigger conditions more complicated. At that point the issue becomes one of optimization: the triggers are expressible, but they execute inefficiently because of lack of sophistication in the trigger processor. Thus, one important challenge becomes efficient monitoring of triggers with complex conditions. There has been only a smattering initial work in the research community along these lines, e.g., [Han92].

Another class of trigger events not yet supported in products—despite their prominence in several research prototypes—is *time-based events*. Although there are numerous interfaces for specifying and scheduling activities based on time, including database activities, to date we have not seen time-based events incorporated directly into a commercial DBMS trigger system.

Product limitations

Product limitations are being lifted, slowly. The limit on number of triggers per event type has been eliminated in most products, since the limitation was an obstacle even to simple trigger-based services. However, most DBMS's still do not integrate their trigger and constraint systems well [CPM96]. Furthermore, run-time behavior may be unnecessarily restricted (e.g., limits on number of rule activations), and many systems still lack a means of prioritizing when multiple triggers are activated at the same time [ACL91]. All of these issues have been addressed in some detail in the research community [WC96], but not yet adopted in all products.

Uniformity

We expect lack of uniformity to improve somewhat as the SQL-99 standard settles in, although standards never seem to solve uniformity problems fully. For example, the different transactional models supported by different DBMS products can have subtle but significant effects on trigger behavior, even if the semantics of the triggers themselves appear identical.

Subtle behavior

There has been little improvement since 1995, except perhaps in understanding the extent of the subtleties [CPM96]. Even in the presence of well-defined trigger semantics, behavior can be surprising. For example, row-level triggers are activated once for each modified (inserted, deleted, or updated) tuple, but no triggers are activated until the modification statement is complete. Thus, the execution of a row-level trigger effectively enumerates through the modified rows in an undefined order, possibly invoking complex procedural logic (and even database updates) for each row. As another example, if triggers invoke external actions, there is no way for the external actions to know if the triggering transaction committed. Thus, an external action may be invoked multiple times for restarted transactions, and it may perform actions based on changes that do not commit. “Deferred” or “commit” triggers are not supported by most DBMS trigger systems, so programming correct deferred behavior requires a significant amount of effort.

These issues are just examples, but they serve to further underscore the importance of the application development and trigger analysis tools, mentioned next.

Development support

Lack of a support environment for developing handcrafted trigger applications is still a significant problem. Although the research community has produced a number of nice theoretical results and prototype implementations in this area, commercial database systems are void of trigger design support, and introducing such tools may never reach the radar screen for database vendors. Thus, instead of the emergence of fully general trigger analysis and design tools, we see the growth in:

- Automatic generation of trigger sets from declarative specifications, as discussed throughout this paper, and already used in a surprising number of trigger applications (second row of Figure 1).
- An intermediate approach between fully automatic trigger generation and completely handcrafted trigger sets: trigger programming "wizards" as described in Section 6.2 that assist in developing correct trigger sets for a particular application.

Performance

Trigger performance can certainly be a problem, although not uniformly across all applications. There are many cases when a small number of triggers embedded in the DBMS clearly outperforms an approach that hard-codes the same functionality into multiple applications. However, when an application permits a very large number of triggers on the same table (e.g., an alerting system), or when trigger conditions are complex as discussed earlier, performance quickly deteriorates in all deployed trigger systems that we know of. (For example, as mentioned earlier, implementing a rule-based inferencing system using database triggers is infeasible at this time given existing trigger implementations.) Again, there has been some work in the research community, particularly in terms of scaling the number of triggers while maintaining good performance, that has not yet found its way into products.

10 Conclusions

Triggers are being used in a variety of significant ways in today's database systems and applications. The primary use of triggers is still to enforce various integrity constraints driven by the application, and we have argued that more uses of triggers than one might think are in fact maintaining constraints of one sort or another. Meanwhile, there is still work to do in trigger processing for researchers and developers alike, particularly to address performance issues and the lack of application development tools.

Acknowledgements

We are grateful to Jim Gray and Ron Soukop for some initial information on trigger applications to get this paper rolling, to Mike Stonebraker for a useful discussion near the finish line, to Richard Sidle for comments on a draft, and to the following folks at IBM for helpful answers to numerous questions: Qi Cheng, Patrick Dantressangle, Stefan Dessloch, Jing-Song Jang, Beth Hamel, Madhu Kochar, Nelson Mattos, and Calisto Zuzarte.

Stefano and Jennifer are grateful to IBM Almaden for providing a stimulating environment for their joint work in active databases, and to Bruce Lindsay, Hamid Pirahesh, and especially Bobbie Cochrane for all their great work in making constraints and triggers a reality in practice (and in helping us assemble this paper).

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.

[BBC+97] P. Bernstein, M. Brodie, S. Ceri, et al. The Asilomar report on database research. *SIGMOD Record*, 27(4):74–80, December 1997.

[BDD+98] R. Bello, K. Dias, A. Downing, et al. Materialized views in Oracle. In *Proceedings of the Twenty-Fourth International Conference on Very Large Data Bases*, pages 659–664, New York, New York, August 1998.

[CF97] S. Ceri and P. Fraternali. *Designing Database Applications with Objects and Rules: The IDEA Methodology*. Addison-Wesley, 1997.

[CPM96] R.J. Cochrane, H. Pirahesh, and N.M. Mattos. Integrating triggers and declarative constraints in SQL database sytems. In *Proceedings of the Twenty-Second International Conference on Very Large Data Bases*, pages 567–578, Mumbai, India, September 1996.

[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.

[CW93] S. Ceri and J. Widom. Managing semantic heterogeneity with production rules and persistent queues. In *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, pages 108–119, Dublin, Ireland, August 1993.

[DHL90] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 204–214, Atlantic City, New Jersey, May 1990.

[EC75] K.P. Eswaran and D.D. Chamberlin. Functional specifications of a subsystem for data base integrity. In *Proceedings of the First International Conference on Very Large Data Bases*, pages 48–67, Framingham, Massachusetts, September 1975.

[Esw76] K.P. Eswaran. Specifications, implementations and interactions of a trigger subsystem in an integrated database system. IBM Research Report RJ 1820, IBM San Jose Research Laboratory, San Jose, California, August 1976.

[Han92] E.N. Hanson. Rule condition testing and action execution in Ariel. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–58, San Diego, California, June 1992.

[HM75] M. Hammer and D. McLeod. Semantic integrity in a relational data base system. In *Proceedings of the First International Conference on Very Large Data Bases*, pages 25–47, Framingham, Massachusetts, September 1975.

[Hor92] B. Horowitz. A run-time execution model for referential integrity maintenance. In *Proceedings of the Eighth International Conference on Data Engineering*, pages 548–556, Phoenix, Arizona, February 1992.

[Inf00] Informix Software. *Informix Media360*, 2000. Available at http://www.informix.com/media360.

[LLPS91] G.M. Lohman, B. Lindsay, H. Pirahesh, and K.B. Schiefer. Extensions to Starburst: Objects, types, functions, and rules. *Communications of the ACM*, 34(10):94–109, October 1991.

[LSPC00] W. Lehner, R. Sidle, H. Pirahesh, and R.J. Cochrane. Maintenance of automatic summary tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 512–513, Dallas, Texas, May 2000.

[Ora00] Oracle Corporation. *Oracle Workflow Technical Configuration in Oracle Applications Release 11*, 2000. Available at http://www.oracle.com/support/library/supportnews/html/workflow.html.

[Ros97] R.G. Ross. *The Business Rule Book: Classifying, Defining and Modeling Rules*. Business Rule Solutions Inc., February 1997.

[SK91] M. Stonebraker and G. Kemnitz. The POSTGRES next-generation database management system. *Communications of the ACM*, 34(10):78–92, October 1991.

[SKD95] E. Simon and A. Kotz-Dittrich. Promises and realities of active database systems. In *Proceedings of the Twenty-First International Conference on Very Large Data Bases*, pages 642–653, Zürich, Switzerland, September 1995.

[SRL+90] M. Stonebraker, L.A. Rowe, B.G. Lindsay, J. Gray, M.J. Carey, M.L. Brodie, P.A. Bernstein, and D. Beech. Third-generation database system manifesto – The committee for advanced DBMS function. *SIGMOD Record*, 19(3):31–44, September 1990.

[Sto75] M. Stonebraker. Implementation of integrity constraints and views by query modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 65–78, San Jose, California, May 1975.

[Tho97] C. Thompson. Database replication. *DBMS Magazine*, 10(5), May 1997.

[UW97] J.D. Ullman and J. Widom. *A First Course in Database Systems*. Prentice Hall, Upper Saddle River, New Jersey, 1997.

[WC96] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, California, 1996.