BONUS OBJECT BUYER'S GUIDE p.25 • MODELING & DESIGN WITH JAVA p.59

JOURNAL OF
OBJECT-ORIENTED Programming

A SIGS Publication
May 1997
Volume 10, No. 2
US $9/Can $12/Intl $13

Database Development

- The Benefits of Triggers
- New Approaches for Merging Object and Relational Technologies

PLUS

Automating Design Pattern Applications

How to Fuse Methods for Software Development

Understanding the OPEN Methodology

JAMES RUMBAUGH
The keys to modeling

ANDREW KOENIG
An interface for examining directories

PUGH & LALONDE
Smalltalk and Internet/ intranet applications

Get ready for Object Expo / Java Expo p.9

SIGS PUBLICATIONS
75624 05
0 248186 9

David W. McKeown & Hossein Saiedian

Department of Computer Science,
University of Nebraska at Omaha
Omaha, NE 68182-0500,
hossein@cs.unomaha.edu

Triggers for object-oriented database systems

Abstract

Triggers are a special form of stored procedure that are automatically executed by a database-management system when specific conditions concerning data arise. Triggers can be used to maintain referential integrity, maintain derived or redundant data, ensure sequence rules, maintain mutually exclusive and inclusive rules, maintain data-dependent values, and access domain-restriction data. The implementation of triggers is an active database concept that recently has been incorporated into many commercial relational databases. As relational database manufacturers augment their databases with object-oriented capabilities, the concept of triggers in an object-oriented database environment raises concern. Some object-oriented purists believe triggers violate the property of object encapsulation, since triggers can directly alter an object's state rather than using the object's methods. Additionally, it is believed that all of the tasks that triggers are traditionally used for can be accomplished by object methods. This article counters the aforementioned beliefs and describes the benefits that triggers will provide to object-oriented databases.

Triggers originated in and are the heart of active database management systems, i.e., systems that can modify data or carry out actions without being directly requested to do so by users or external application programs. This can be contrasted with traditional database management systems (DBMS) that are passive and execute queries or transactions only when explicitly requested to do so by a user or an application program.[13] Many terms may be used synonymously for triggers in the DBMS arena; for example, production or forward-chaining rules, event-condition-action (ECA) rules, situation-action rules, monitors, or alerters. Triggers, or more precisely DBMS trigger subsystems, allow users to define conditions or invariants on the data in the DBMS and specify actions to be taken if events occur that result in a violation of one or more of the specified conditions. For example, a user of a trigger-capable DBMS could define a trigger on an inventory table that sends a reorder notification to the purchasing department when an inventory item drops below a certain threshold. Many relational DBMSs vendors have enhanced their systems by incorporating triggers into their products. The decision to implement triggers was an easy decision for relational DBMS vendors to make because triggers provided active, real-time monitoring capabilities to their systems—features today's sophisticated DBMS users demand. However, the decision to implement triggers in object-oriented database systems (OODBMS) is not without opposition. OODBMSs allow users to define both the structure of complex objects and the operations (called methods) that can be applied to these objects. One of the principal qualities of any object-oriented system is encapsulation, which requires an object's state (called attributes) should only be altered by the object's methods. Many object-oriented purists view the implementation of triggers in an OODBMS to be a violation of the encapsulation principal, since the actions that triggers might take involve direct manipulation of the object's data rather than accessing the data through the object's methods. The purpose of this article is to refute this notion by presenting a case for implementing triggers in an OODBMS.

Although the argument against the implementation of triggers may at first glance seem quite reasonable, an in-depth study into the topic must be conducted before making a final decision. An examination of how triggering systems are implemented must be conducted. The benefits and drawbacks of triggering systems must be determined without respect to the encapsulation principal. If triggers provide functionality to DBMS users that cannot be provided by OODBMS methods, it must be documented. The benefits and drawbacks of enforcing strict encapsulation in an OODBMS must be examined. The effects of implementing triggers in the OODBMS must be explored with respect to encapsulation. Other breaches of encapsulation in OODBMSs and techniques for dealing with the violation have to be investigated. This article seeks to address all of these concerns and recommend a course of action concerning the implementation of triggers in OODBMSs based on the findings.

Triggers

TRIGGERS Triggers have been defined many ways in the DBMS literature. Graham[7] defines triggers as rules that fire in response to a change of state. The same literature further defines triggers to be procedures that are attached to data structures and that fire when the structure is accessed. Triggers are special cases of methods that are activated by changes to the state of the data rather than by user or application-program requests. Bertino and Martino[3] define triggers as actions that are automatically executed by the system when specific conditions concerning data arise. Sybase Inc.[12] defines triggers as a special form of stored procedure that goes into effect when a user gives a change command, such as insert, delete, or update to a specified table or column. Owens and Adams[11] define triggers as actions to be taken whenever an SQL statement affects a table and, equally significant, at a specified time or state within the SQL statement's life cycle. Kotz, Dittrich, and Mulle[8] define triggers as a special concept that integrates "reactions" into the DBMS. Compared to mere actions, a reaction additionally encompasses the cause for and/or the timing of the action. Trigger concepts in the DBMS arena allow for database operations that, conditionally or unconditionally, succeed other database operations.

The decision to implement triggers in object-oriented database systems is not without opposition.

By carefully examining the previous definitions of triggers, a more concise definition of triggers can be constructed: Triggers are conditions defined on data in a DBMS (of any variety) that, when satisfied, initiate arbitrary actions (perhaps in terms of stored procedure) to be carried out. This is the definition that will be used for the purposes of this article.

TRIGGER IMPLEMENTATION Typically, triggers are implemented in a separate subsystem of the DBMS. The trigger subsystem is generally a component of a larger rule-processing subsystem. Much like a query-processing subsystem in a DBMS, the trigger/rule subsystem consists of one or more processes that run constantly in the system as demons. Demons are necessary to allow real-time monitoring and processing of rules. These demons detect when certain events have occurred, evaluate conditions, and initiate arbitrary actions associated with the conditions. Additionally, the trigger/rule subsystem allows users and application programs to maintain rules. Maintaining rules involves defining, modifying, activating, deactivating, and deleting rules and their corresponding stored procedures.[6]

A number of techniques exist for the implementation of triggers in DBMSs. A very simplistic approach is being taken by many commercial relational DBMS manufacturers, such as Sybase and Oracle, for implementing triggers.[11,12] The implementers of this technique realize that database modification transactions (insertions, updates, and deletions) are the means by which the database is taken from one consistent state to another. Therefore, this technique provides for condition checking and triggered actions only when database modification events occur. As this article will further discuss, this simplistic technique furnishes only a subset of the active database capabilities possible through triggers.

At the conceptual heart of triggers and active databases is the event-condition-action (ECA) model. Several other models, such as the rule triggering system model[14] and the event/trigger mechanism,[8] can be thought of as being synonymous to the ECA model and will be for the purposes of this article. Triggers and active databases are centered around the notion of rules. Rules are defined by users, applications, or database administrators and specify desired active behaviors. In their most abstract form, rules consist of three parts: events that cause the rule to be triggered, conditions that are checked when the rule is triggered, and actions that are executed when the rule is triggered and its condition is true. From the three components of a rule, the ECA model emerged. A thorough summary of each of the three components follows.

The first component of the ECA model is the event. As stated earlier, the event specifies what causes the rule to be triggered. There are many useful triggering events that have been discussed in literature and implemented individually in one or more DBMSs, such as data-modification, data-retrieval, temporal, and application-defined events. Data-modification events might be specified as one of the three SQL data-modification operations—insert, delete, or update—on a particular table. In an OODBMS, a modification event might be specified as the creation, deletion, or modification of a particular object. It might also be specified as the execution of a particular object method that modifies the state of the object. Data-retrieval events can be specified as those operations that query a table in a relational DBMS, fetch an object in an OODBMS, or extract data from an object in an OODBMS. Temporal events specify that a rule should be triggered at an absolute time, at a repeated time, or at periodic intervals. Application-defined events allow an application to declare an external event (e.g., a fire alarm being pulled) that occurs outside of the DBMS. In addition to the above events, the combination of single events into composite events and the ability for events to pass parameters to the rule's condition and/or action would be both extremely desirable features of any event-monitoring subsystem.

Conditions specify a database state to be checked once the rule is triggered and before the action is executed. Useful conditions in a DBMS include database predicates, database queries, and application procedures. Database predicate conditions specify that certain predicates must hold true in the database. For example, a condition expressed as a predicate might state that the salary for an employee must never exceed the salary for the man-

ager of a department. Conditions might also be specified as a query that is to be executed and evaluated using the DBMS' query language. For example, a condition specified as a query might retrieve all data with value below a certain threshold. The meaning can either be that the condition is true if and only if the query produces an empty answer, or that the condition is true if and only if the query produces a nonempty answer. Lastly, a rule condition might be specified as a call to a procedure written in an application programming language (e.g., limit-exceeded()), where the procedure may or may not access the database.

Actions are executed when rules are triggered and a rule's condition evaluates to true. Actions useful in a DBMS fall into four categories: data-modification operations, data-retrieval operations, other database commands, and application procedures. In active relational DBMS modification, actions correspond to insert, delete, or update operations. In an object-oriented active DBMS modification, actions might be object creation, object deletion, or method calls that modify objects. Data-retrieval actions are basically queries on the DBMS. A relational DBMS might allow such rule actions to specify SQL select operations, whereas an object-oriented DBMS might allow rule actions to specify object fetches, or method calls that retrieve objects. A rule action might allow any database operation at all to be specified. Examples of some other DBMS operations that might be considered as useful actions include operations for data definition, operations for transaction control (e.g., commit), and operations for granting and revoking privileges. Lastly, application procedures can be specified as rule actions. As such, a rule action might be specified as a call to a procedure written in an application programming language, where the procedure may or may not access the database.[10,13]

APPLICATIONS OF TRIGGERS Triggers are used to ensure uniqueness or primary key constraints. If a tuple has a field or several fields that must be unique or act as a primary key, a trigger can be established that will ensure that inserted and updated values are unique. Triggers are used to maintain the referential integrity of tables with corresponding foreign tables. Triggers are used in this scenario to disallow the insertion or modification of a foreign key value with no corresponding object in the foreign table. Additionally, instead of disallowing the operation, a trigger could be specified that would automatically create a default tuple in the foreign table for the foreign key. Because a value corresponding to the original foreign key would then exist in the foreign key, referential integrity is ensured. Triggers are used to maintain not-null constraints. If a field must not contain a null value, a trigger could disallow the insertion or updating of that field with a corresponding null value. Additionally, rather than disallowing the operation, the trigger could simply populate the field with a non-null default value.

Triggers are used to maintain derived (data dependent) and/or redundant data. Derived data is data obtained by some sort of manipulation of other data in the database. For instance, any time the salary of an employee is changed in the employee table, it may be desirable to automatically update the salary expense accordingly in

the expenses table. Redundant data is data maintained in more than one table. For example, an individual can be both an employee and a student at a university. If the person's address changes in the student table it would be desirable to automatically update the person's address in the employee table and vice versa.

Triggers are used to maintain mutually exclusive and inclusive rules. Mutual exclusion specifies that if data values occur in one table they must not appear in another table and vice versa. Mutual inclusion specifies that data values that occur in one table must appear in another specified table.

Triggers are used to ensure sequence rules. In other words, triggers are used to insure that certain database transactions occur in a specific well-defined order. Recent developments in active databases have even begun to define temporal triggers and triggers on database histories. Temporal triggers are triggers based on time. As such, users can define events that occur at a specific date and time or at specified intervals. Additionally, time-based sequencings of actions are possible. Triggers on database histories allow the monitoring of the database log file for specific conditions. For example, a trigger could monitor the history for an extraordinary amount of aborts over a given period of time and alert the database administrator of the condition.

Triggers are used to ensure uniqueness or primary key constraints.

Although the number of uses for triggers in a relational or active database system is quite large, the ways in which triggers may be used should not be limited to those listed here. Because the ECA model underpins the implementation of triggers and even external events and actions can be defined for triggers, conceptually, the number of uses for triggers is limitless. Any event that can be made to send a signal to the trigger subsystem demon can be utilized in the model. Likewise, any stored procedure, written in a data manipulation language (DML) or an external programming language, can be used as an action. A careful examination of how triggers could be used in an OODBMS will be postponed until after the characteristics of an OODBMS have been discussed.

OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS Object-oriented databases arose out of a need to persistently store complex objects created by object-oriented programming languages, the need for new data types for storing images or large textual items, and the need to define nonstandard application-specified operations. The object-oriented approach offered the flexibility to handle these requirements without being limited by the data types and query languages of traditional DBMSs. The key feature of object-oriented databases is the ability to define complex objects and the operations that can be applied to these objects.

Until recently, it was very difficult to describe the exact characteristics a DBMS must possess to be classified as an OODBMS because no agreed upon object database standard existed. In 1994, the Object Database Management Group (ODMG) published the Object Database Standard, ODMG-93, which was an attempt by industry leaders in the OODBMS arena to agree on a standard with which all OODBMSs must comply.[4] The ODMG-93 Standard defined the object model to be used by compliant OODBMSs. The object model is slightly different from the Object Management Group's (OMG) object model, but still compatible. Additionally, the ODMG-93 standard elucidated a grammar for defining objects called the Object Definition Language (ODL), a language for querying OODBMSs called the Object Query Language (OQL), C++ and Smalltalk language bindings to OODBMSs, and methods for OODBMSs to communicate with CORBA.*

Even though the specifics of how an OODBMS should be implemented are not agreed upon, a high-level conceptual standard of OODBMSs has been established by Atkinson *et al.*-[1] "Object-Oriented Database Manifesto" was presented at the First International Conference on Deductive and Object-Oriented Databases. It defines 13 commandments with which an OODBMS must conform. Eight of the commandments are rules that make an OODBMS an object-oriented system. Five of the commandments are rules that make an OODBMS a DBMS. The commandments that make an OODBMS an object-oriented system are as follows:

- Commandment 1: The system must support complex objects.
- Commandment 2: Object identity must be supported.
- Commandment 3: Objects must be encapsulated.
- Commandment 4: The system must support types or classes.
- Commandment 5: The system must support inheritance.
- Commandment 6: The system must avoid premature binding.
- Commandment 7: The system must be computationally complete.
- Commandment 8: The system must be extensible.

The five commandments that make an OODBMS a DBMS are as follows:

- Commandment 9: The system must be able to remember data locations.

* Unfortunately for the developers of the ODMG standard, at the time they were developing ODMG-93, ANSI was hard at work extending the SQL-2 standard to work with OODBMSs. The new SQL-3 standard provides its own object model, object definition language, and object query language. For the most part, the ODMG-93 standard and the SQL-3 standard are compatible, although not completely. It is hoped that in the future the two standards can be merged. Until such a time, the OODBMS world still is without a definitive standard.

- Commandment 10: The system must be able to manage very large databases.
- Commandment 11: The system must accept concurrent users.
- Commandment 12: The system must be able to recover from various failures.
- Commandment 13: The data query must be simple.

Of the 13 commandments listed, three (3, 7, and 13) are particularly relevant to this article and will therefore be discussed further. Commandment 3 states that objects in the OODBMS must be encapsulated. Encapsulation means that the objects must have a *public* interface, but *private* implementation of data and methods. The encapsulation feature ensures that only the public aspect of the object is seen, while the implementation details are hidden. It is this principle that adding triggers to OODBMSs will purportedly violate. Commandment 7 states the system must be computationally complete. This capability is provided by augmenting the data manipulation language (DML) of the DBMS with the features of modern programming languages. The resulting DML should allow any type of operation to be expressed. Because of the expressive power of the DML, many believe triggers should not be implemented in OODBMSs, but instead handled by the object's methods, thus preserving encapsulation. Lastly, Commandment 13 states that data queries must be simple. Efficient querying is one of the most important features of any DBMS. Relational DBMSs have provided a standard database query method through SQL, and OODBMSs must provide similar query capabilities and efficiencies.

A knowledge of how OODBMSs are used is essential to making the trigger implementation decision. Object-oriented databases have been characterized as "next-generation" database management systems for advanced applications. Traditional usages of OODBMSs have included computer-aided design (CAD), computer-aided manufacturing (CAM), computer-aided software engineering (CASE), and intelligent offices, which include office automation and document imaging. CAD denotes the use of computerized systems and tools for designing products. Through the direct representation of the hierarchical structures of complex design objects, the support of versioning when designs are altered, and the control of concurrent accesses and updates in projects, object-oriented databases provide ideal persistent repositories for CAD objects. CAM refers to a software system that offers assistance in the manufacturing or production of components or machines. With CAM, computer systems are involved in monitoring and controlling the production cycle. This means that computers operate the manufacturing floor. The status of various machines and monitors is continually processed and communicated by the system. The role of the underlying object-oriented database system is the storage of objects, the object states, and the history of the object states in the manufacturing process. In CASE tools, OODBMSs are used to store and retrieve the code base of complex software-engineering projects. The various libraries that are used to construct the system can be modeled as complex objects. The OODBMS is also used for versioning and provides a natural check-out/check-in mechanism through object locking. In intelligent office environments, OODBMSs are used to persistently store a variety of complex objects such as faxes, scanned images, multimedia, and voice data.[12]

TRIGGERS IN AN OBJECT-ORIENTED DBMS

Not every use of triggers specified for relational or active databases will map to OODBMSs. Additionally, OODBMSs may have their own special needs that triggers could support quite well. This section seeks to examine which uses of triggers previously mentioned are applicable to OODBMSs and identify any uses of triggers that are unique to OODBMSs.

Objects in OODBMSs do not typically have keys, but rather a unique object identity; therefore, triggers would not be used to ensure primary key constraints. The object identity's uniqueness is guaranteed by the system itself. Uniqueness constraints would be less likely to be needed in an OODBMS, but are not entirely unthinkable. Triggers could be used to enforce uniqueness constraints. Referential integrity is guaranteed in an OODBMS because references to other objects are automatically handled by the formation of complex objects. All references are a part of this complex object and therefore maintained automatically. Triggers could be used to maintain "not null" constraints in an OODBMS in the same manner previously mentioned for a relational DBMS. Triggers can still be used to

maintain derived, redundant data, mutually exclusive rules, mutually inclusive rules, and sequencing rules in an OODBMS. In an OODBMS, temporal triggers and triggers on database histories are even more desirable considering the nature of the applications in which OODBMSs are typically used.

Of great applicability in an OODBMS, triggers can be used to enforce complex semantic rules. Nonstandard applications like CAD/CAM, image processing, CASE, and intelligent offices require database systems with facilities to handle sophisticated semantics. Advanced data models, such as OODBMSs, supporting complexly structured objects, abstract data types, and such have been developed to this end. However, any data model remains restricted to static and global semantics, not taking into account individual and dynamically changing issues. It is therefore necessary to create additional semantic rules, which may be rather complex and possess a variety of checking as well as enforcement requirements.[8]

From the previous discussion, it is readily apparent that triggers could be used to satisfy a variety of OODBMS needs. The question that remains to be answered is should they be used even though they violate the encapsulation principle? The next section of this article will examine this issue in greater detail.

DISCUSSIONS The following section of the article takes a critical look at arguments both for and against the implementation of triggers in an OODBMS. First, the concept of supporting encapsulation through the encoding of rules into object methods is explored. Next, the effects of violating encapsulation are discussed. Then, the violation of encapsulation by query languages is exposed. Lastly, methods of implementing triggers in OODBMSs are explored.

SUPPORTING ENCAPSULATION THROUGH OBJECT METHODS Opponents of OODBMS triggers not only believe that triggers violate encapsulation, they also believe that properly coded methods can be used to provide the same functionality as triggers without violating encapsulation. One of the 13 OODBMS commandments was that the DML be computationally complete. Does computational completeness mean that demon methods can be created that will monitor the state of the object and trigger actions? Probably not at this time. However, if the DML is augmented to use procedures compiled in a programming language, such as C++ directly, then demon processes would be possible. However, one must consider the consequences of coding one or more demons for each object in the system. Millions of demons could be required to check constraints, ensure integrity, guarantee semantic rules, etc. This would obviously be inefficient compared to a single, central demon maintaining all the rules of the system. Along the same lines as the demons, it is uncertain how the DML would handle temporal events. Because of the static nature of the objects in an OODBMS, it is uncertain how demons for maintaining rules or handling temporal events would get instantiated. Object methods do not start themselves automatically in an OODBMS, but rather are invoked by users or application programs. Would a user have to instantiate all the demon methods? This is not clear.

Certainly all object constraints and rules could be checked for in the object's methods. However, problems exist with embedding constraints and rule enforcement in the methods. Every method that modifies an attribute that needs to have a rule or constraint enforced must include the logic for enforcing the constraint. If an object has 20 different methods that can update an attribute on which a rule must hold, all 20 methods must encode the same logic. This has implications in the area of maintainability and efficiency. The ability to create new rules, delete existing ones, and toggle rules on and off instantaneously is lacking from this method as well. Another area of concern is that the DML for rule or constraint enforcement could become quite complex, especially when considering events that must occur in a given sequence. Much coding could be required of the system developers. Similarly, if the code for enforcing the rules is overly complex it can be difficult to extract the actual description of what exactly is being enforced. Having stated all the above, it is still possible to enforce rules and take actions through an object's methods, however inefficient and unmanageable the actual implementation.

TRIGGER EFFECTS ON ENCAPSULATION Encapsulation is a desired property of object-oriented systems that seeks to maintain the integrity of objects by allowing the object's state to be altered only by the object's methods. State variables of the object are not visible to users. Only the object's methods are visible. In an OODBMS, what are the implications of not enforcing encapsulation? Other objects, users, or applications could incorrectly alter the state variables of the object. Couldn't the state variables be incorrectly altered by using the object's methods? The answer is yes and no. The answer is yes if the methods do no rule or constraint checking before making the alteration. The answer is no if the methods check rule violations. The following two statements sum up the importance of encapsulation in an OODBMS: If rule checking is built into the methods and the methods are used religiously, encapsulation ensures the consistency of the object. If no rule checking is built into the methods or the methods aren't always used, encapsulation provides not much more than an indirect means to alter the object's state without ensuring consistency. These statements could almost lead one to believe that the opponents to triggers are correct in thinking that trigger-like conditions and actions should be programmed into the methods. However, one of the uses of triggers is to maintain consistency in the database, regardless of whether modifications are done directly or through methods. Even if encapsulation is violated when updating data in the database, triggers will be there to ensure consistency. The same cannot be said of object methods. In essence, by saying that triggers cannot be implemented in an OODBMS because they violate encapsulation, we are actually saying we cannot implement a method of ensuring object consistency because it might violate the consistency of the object!

TRIGGERS AND QUERY PROCESSING ENCAPSULATION Database users expect OODBMSs to support queries. They expect the queries to be handled quickly and efficiently. They also expect to have a variety of features avail-

able to them, such as aggregate functions, sorting and grouping functions, and nested queries.[9] Designers of the first OODBMSs quickly discovered that without violating encapsulation, queries would be very slow and inefficient. Additionally, advanced features would be extremely difficult to implement. If the query language cannot violate encapsulation, then basically the only access to the objects is through methods, and the physical data is invisible. If we want to see the data, we are obliged to violate encapsulation. DBMSs provide very fast low-level methods for finding, retrieving, joining, and sorting data. The use of methods would negate the ability of the DBMS to work with large amounts of data at a low level. From this example, it is apparent that it is acceptable to violate encapsulation in some circumstances.[2]

Of three articles found closely relating to the implementation of triggers in OODBMSs, all three seem to have taken a common approach. The approach was to represent constraints and rules semantically and store the semantic definitions as full-fledged objects in the OODBMS. The OODBMS would automatically generate the appropriate low-level implementation of the rule/trigger. An active rule/trigger subsystem constantly monitors system events, evaluates conditions, and calls stored procedures (actions) when necessary.[5,6] The actions invoked by the trigger subsystem can be calls to object methods or directly alter the object. Calls to object methods, rather than directly altering or querying the data, would ensure encapsulation.

CONCLUSIONS Triggers could be utilized for a variety of tasks in an OODBMS, ranging from constraint enforcement to initiating timed events. Because of the way triggers directly alter the data in tables in relational and active DBMSs, it is believed that the same would be true in OODBMSs, thus violating the principle of object encapsulation. Coding rules into the object's methods was suggested as a solution to the problem. However, there are a variety of problems that can arise from this approach, some worse than violating encapsulation. Encapsulation is a principle that seems to be selectively enforced in OODBMSs. Exemptions must be made to encapsulation wherever practical. Triggers are one area where the benefits of relaxing the encapsulation rule outweigh the effects of violating encapsulation. After viewing several implementations of triggering systems in OODBMSs in which rules are stored as full-fledged objects in the system, it can be argued that such a system allows for interobject communication and does not violate encapsulation. Regardless of whether one views triggers as violating encapsulation, they should be implemented in OODBMSs. Future work in this area must produce a common, consistent grammar for representing triggers and rules in OODBMSs, much like the ODL for defining objects. ■

References

1. Atkinson, M., *et al.*, "The Object-Oriented Database Manifesto," in *Deductive and Object-Oriented Databases*, W. Kim, J. M. Nicolas, and S. Nishio, Eds. Elsevier Science Publishers, New York, 1990.
2. Bancilhon, F., C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System: The Story of O2*. Morgan-Kaufmann, San Francisco, 1992.
3. Bertino, E. and L. Martino. *Object-Oriented Database Systems: Concepts and Architectures*. Addison-Wesley, Reading, MA, 1993.
4. Cattell, R.G.G. *Object Database Standard: ODMG-93, Release 1.1*. Morgan Kaufmann, San Francisco, 1994.
5. Chakravarthy, U. S., and S. Nesson. "Making an Object-Oriented DBMS Active: Design, Implementation, and Evaluation of a Prototype," in *Proc. International Conf. on EDBT*, Venice, March 1990.
6. Dayal, U., A. Buchmann, and D. McCarthy. "Rules are objects too: A knowledge model for an active, object-oriented database system," *Advances in Object-Oriented Database Systems*, pp. 129–143, Sept. 1988.
7. Graham, I. *Object-Oriented Methods*. Addison-Wesley, Reading, MA, 1991.
8. Kotz, A., K. Dittrich, and J. Mulle. "Supporting Semantic Rules by a Generalized Event/Trigger Mechanism," in *Lecture Notes in Computer Science*, J. Schmidt, S. Ceri, and M. Missikoff, Eds. Springer-Verlag, New York, 1988: 76–91.
9. Loomis, M. E. S. *Object Databases: The Essentials*. Addison-Wesley, Reading, MA, 1995.
10. McCarthy, D. R. and U. Dayal. "The architecture of an active database management system," in *Proc. 1989 ACM SIGMOD International Conf. on the Management of Data*, pp. 215–224, June 1989.
11. Owens, K., and S. Adams. "Oracle 7 triggers: The challenge of mutating tables," *Database Programming and Design*, pp. 47–54, Oct. 1994.
12. Sybase Inc. *Transact-SQL User's Guide*, 1987.
13. Widom, J., and C. Stefano. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, 1995.
14. Zhou, Y., and M. Hsu. "A theory for rule triggering systems," in *Lecture Notes in Computer Science*, F. Bancilhon, C. Thanos, and D. Tsichritzis, Eds. Springer-Verlag, New York, 1990: 407–421.