ENHANCING INTEGRITY BY INTEGRATING BUSINESS RULES, TRIGGERS, AND ACTIVE DATABASE TECHNIQUES

David H. Olsen (dolsen@b202.usu.edu) Utah State University
Olga Yatsenko Utah State University

ABSTRACT

Integrity, in database terms, is the quality of trustworthiness that the data in a database is imbued with; high levels of integrity reassure all users of a database system that they can rely on the information they retrieve from the system. It is axiomatic that integrity is crucial to accountants that use and audit such systems because no one would use or rely on a suspect accounting system. One of the most interesting types of integrity concerns is adherence to business rules; those standard operating procedures and policies that dictate how an organization conducts transactions in its business environment. While such rules can be enforced in programs that use the database, if the business rules are implemented within the database, they need not be recreated when new applications are developed but will be automatically enforced for all applications. This practice is ever more widely adopted both because it saves programmers' wages and because it inherently increases the overall integrity of systems adopting it. Two mechanisms for enforcement of business rules at the database level are constraints and triggers. These may be of particular interest and importance to auditors who, given that they are confident of their understanding of the function of such mechanisms, may simply review the triggers and constraints and possibly decide to place greater reliance on the system itself, saving auditor time and increasing the quality of the audit. This paper explains and illustrates trigger and constraint design approaches so as to inform the auditor and facilitate audits involving such systems.

Keywords: Triggers, active databases, database constraints, business rules, database design

Introduction

Business rules will be defined more precisely in a subsequent section of the paper but for now may be thought of as those standard operating procedures and policies that dictate how an organization conducts transactions in its business environment. Some business rules are so implicit as to be virtually invisible; in a manual system it is just common sense, for example, that a customer on an order form be a customer of the company (likewise the salesperson be an employee and the products sold be part of the company's inventory.) Other rules are more explicit; e.g., an organization's credit granting policy.

In the migration from manual to automated systems several approaches to dealing with business rules have presented themselves. Some implementations did nothing, i.e., employed the same manual enforcement of policies and procedures as had been done in the manual systems that proceeded them. Some adopted the limited built in capacities of the early database management systems they were built on (abbreviated DBMS; when we speak here of DBMS's we are referring to software that manages data built on the relational database model, one of several logical models of how programmers think of data as being stored and the dominant model currently employed in business around the world today.) This approach supported checking for correct data types for entered data (e.g. that a date entered be a legitimate date, that a cost figure be a

valid numeric figure perhaps within pre-approved range.) This approach encouraged accuracy in data input and to that extent enhanced integrity but did little to support an organization's explicit business rules. A third class of approaches, and the one that most concerns this paper is the active database approach.

Prior to the implementation of active database techniques, if system implementers wanted a system to enforce business rules, programming code had to be written into business applications; this was labor intensive and had to be redone for each new application that came online. Active database techniques have been used in recent years to improve quality in areas such as manufacturing and finance. These same active database techniques are becoming increasingly common in mainstream business systems because of integrity concerns. In short, an active database monitors events (transactions) that involve the database and initiates actions if desired (a more complete description of active database operations follows.) An active database that enforces crucial business logic with mechanisms such as triggers and constraints has higher levels of integrity than information systems that either enforces business logic at the application level, or systems that do not enforce business rules at all. An understanding of the active database approach, as well as the triggers and constraints that are its typical mechanisms, will allow an auditor or accounting manager to be a well informed active participant in system design and potentially save valuable time and increase the quality of audits conducted on active database-enabled systems.

In this paper, active database techniques are described and some examples are given in the context of a simple business database. The paper provides a review of relevant active database and business rules research and a foundation of salient active database concepts. Techniques for communication of business rule requirements between accounting professionals and programmers are discussed with inclusion of some original approaches. Finally, the paper concludes with a discussion concerning future research in this area.

Active Databases

Traditional relational databases are considered passive in the sense that, other than enforcing primary key uniqueness and perhaps referential integrity constraints, when legitimate data is entered the sole action taken by the DBMS is to store the data in the database. Active database systems provide facilities that monitor and respond to changes of relevance to the database (Dinn 1999). Active databases (and this assumes a DBMS enabled with active mechanisms), may initiate independent actions as a result of the entered data meeting pre-specified conditions. Active databases are very useful for accounting systems because such systems contain many rules and because data is interrelated. Rather than manually applying the rules, an active database automatically and appropriately responds.

An example of maintaining derived data using triggers can be seen in a simple sales database that includes the attributes sales amount, tax amount, and total sales. If the sales amount were $100 and the local tax rate were 6.25%, the tax amount would be $6.25 and the total sale would be $106.25. The tax amount and the total sales are derived data, and as such could simply be calculated at run time. There are however, good reasons to store the data including performance reasons. Given that the derived data must be stored, a trigger is the best method of calculating and inserting the data as it is automatic and no application program would have to be launched. The other types of tasks could be illustrated with similar examples.

In contrast to a passive database system, an active database monitors pre-specified situations and, when the given conditions are met, triggers an appropriate response (Brownston et al., 1985). For example, when sales are made on account, it is axiomatic that some will become bad debts. It is also a well established fact that, as accounts move farther beyond their due date, the probability of collection declines progressively. Typically, an accounts receivable aging list is prepared at specified intervals after which a person in accounting reviews the list to identify specific receivables that are still unpaid after some arbitrary period, often 60 days. These overdue accounts then receive attention, generally a letter from Credit and Collections.

Alternatively, generalized rules can be implemented in an active database system that creates a collection risk report whenever a receivable hits the 60-day mark. An event, the system clock marking a new day, would trigger a rule. The rule would check the condition of all receivables and for those that exceeded 60 days would initiate an action or actions. These include sending an e-mail to the Credit and Collections Manager, generating correspondence to the delinquent customer, and starting a review of the customer's credit history. This might commence with the system generating an event initiating an ECA regarding the credit history of the account, a questionable credit history could execute yet another series of actions.

To summarize, an active database makes use of pre-specified rules that are "triggered" on a given event and the related conditions. Thus, rules can be used to automate certain tasks and also to augment the manual completion of other tasks.

Business Rules

Accounting managers and data administrators often struggle to understand each other's areas. Data administrators are often unaware of business processes and the critical aspects of the business, most especially business rules, being primarily concerned with the technology involved in database implementation. Accounting managers are often unaware of the precise technologies that underlie fundamental systems (and here we are most interested in technologies for enforcing business rules), being concerned far more with the design of and adherence to business rules. Communication is key to resolving such impasses; the need is for a shared vocabulary and some shared concepts.

The conceptual database design process is intended to facilitate better communication between these two groups. Conceptual database design, often referred to as data modeling, is the documenting and communicating of a technical database worker or workers' understanding of an organization's business environment; documenting so as to prepare for implementation and communicating so as to verify with the sources (e.g., the accounting manager) that the understanding is correct. This process often involves examination of existing system artifacts such as forms and reports as well as requirements gathering interviews with many system stakeholders. It is typically an iterative process wherein one pass at documenting, say, business rules leads to a conference where the understanding is verified and / or modified.

Techniques to support this process currently in wide use include the entity / relationship (E/R) diagram, the resources / events / agents (REA) model (McCarthy, 1982), semantic object modeling, object-oriented data modeling, etc. Many of these models are excellent at specific phases of the analysis and design process, but each seems to lack the ability to fully illustrate the

business processes so that effective communication between data administrators and business managers can occur. More specifically, they all are most concerned with the organizations stored data as is appropriate to a traditional passive database. But, if we anticipate that our database is going to, in some circumstances, take active, independent actions, we need to communicate the conceptual business rules that dictate that action as well as what the organization's data is. In other words, we need to explicitly communicate the business rules as well.

A business rule is defined as "a statement that defines or constrains some aspect of business" (Von Halle, 1997). Business rules must relate in obvious ways to important aspects of the accounting process which includes business knowledge and behavior. The concept of the business rule was identified to represent a communication methodology that can bring the goals of accounting managers and data administrators together (Von Halle, 1997).

There are four categories of business rules: definitions, facts, constraints, and derivations (Von Halle, 1997). Definitions are used to delineate different aspects of the business (Von Halle, 1997). For example, a definition of a capital lease could be "a rental agreement that represents, in substance, the purchase of an asset and the incurrence of a liability." A fact is a property or role of an object or a relationship that an object has with another object (Von Halle, 1994). An example of a fact from FASB No. 13 would be "A lease that transfers substantially all of the benefits and risks incident to the ownership of property should be accounted for as the acquisition of an asset and the incurrence of an obligation by the lessee."

A constraint is a condition about the data that must be true. For example, a constraint could be "if a lease is treated as a sale, the lessor should record a sale of the property and recognize a receivable for the future rent." Finally, a derivation is a business rule that is computable from existing information. An example of a derivation would be "the difference between the cost of the property on the books of the lessor and the fair value of the property is recorded as gross profit (loss) on the sale at that time."

By identifying a methodology that data administrators can follow to capture business rules, a line of communication is opened between business managers and data administrators. This occurrence is very important as businesses increasingly rely on information technology to assist in data retrieval, decision making, and business projections. For an information system to assist in a business's success, it is imperative that data administrators understand the goals and requirements of the business. This understanding will give data administrators the ability to design the optimal system for their company. This need to communicate not just the data but also the processing *in the database design*[1] can be seen in the emergence of the Unified Modeling Language (UML) approach which extends E / R diagrams to included methods (processes executable by objects) being documented in textbooks on database design (Kroenke, 2002). We find approaches such as the UML to be less than satisfactory though, owing to the reliance on an object oriented paradigm when the relational model is still far and away the dominant one in business applications and to their implicit rather than explicit treatment of business rules.

[1] Processing requirements are typically documented in systems analysis and design; the design of the database is a subset of this activity and is traditionally data rather than process driven.

Trigger and Constraint Examples

Figure 1 on the next page illustrates the framework / approach recommended in this paper which includes the traditional function of identifying the business requirements. These requirements are the input for the business rules listing to document the processing the active database will perform and either the entity relationship (E/R) diagram, the resources, events, agents (REA) diagram or the class diagram, depending on the approach used by a given organization for documenting data requirements, is the basis for the database schema. Once a business rules listing is compiled, it is the input to an active database in the form of constraints and triggers. Obviously, this two pronged approach will require more effort to perform but we contend that a database system that incorporates the business rules as a part of the database definition inherently has more integrity than systems that enforce rules via an alternate method. Let's examine this contention in more detail.

Three methods of enforcing business rules include manual enforcement, application enforcement, and database enforcement. An example of manual enforcement is simply someone regularly checking customer balances and comparing those to the credit limits. If someone has exceeded their credit limit, they are denied further credit until the outstanding balance is reduced. Obviously, manual enforcement has limitations as labor costs are increased by having a person check balances. Additionally, the timeliness of checking balances is likely compromised as someone might exceed their limit before a manual check is conducted.

Application enforcement automates the process by including the rule enforcement in each application, which might be a JAVA, C++, or Visual Basic program that accesses the database. This is still a problem because new applications may be developed without the rule logic, which could bypass the business rule safeguards. Moreover, a nefarious act could include directly entering data into a table, which bypasses the application safeguards and allows credit orders that would have been disallowed.

Database enforcement is the method we prescribe which includes implementing business logic using constraints or triggers, which are application independent and are more robust.

A trigger is a database mechanism that is "fired" or executed if, upon the occurrence of a certain event, a specific condition is true and may be defined on updates, deletes, or inserts. Constraints are somewhat similar but are used for more simple tasks like enforcing maximum or minimum values or restricting input to just a few values.

Examples of how these concepts are implemented are illustrated in the following figures. Figure 2 shows a structured query language (SQL) data definition language (DDL) listing of the customers and orders tables in ANSI SQL. A DDL is a language that is used to create the structure of a database and an auditor examining a system must, in order to understand the built-in protections an active database might enable, be able to translate the meaning of such a language. SQL is the defacto industry standard for both DDLs and DMLs (Data Manipulation Languages, a language used to populate user data in a database, e.g., inputting customers, orders, etc.) and as such is an excellent candidate for an auditor to work in; it is easily learned and understood. It is not our purpose here to provide a primer in SQL but the annotations in Figures 2 are hopefully easy to understand. Note in Figure 2 that constraints heighten integrity by

prescribing and defining credit limits (it is axiomatic that management wishes to prevent customers from placing credit orders in amounts greater than the predefined limits) and likewise provide increased accountability by not allowing "illegal" customers from placing orders. An auditor versed in SQL could assess the system by analyzing such declarations and by doing so save valuable time and place greater faith in her/his analysis.

Conclusion

Integrating business rules, triggers and constraints into one methodology is superior to other approaches for two reasons. First, the rules are implemented at the database level, which means that regardless of whether existing applications are modified or new ones are created, the rules will be executed. Second, rules may be identified in the business rules document and then mapped to specific triggers and constraints, which enforce those rules. Both of these reasons contribute to the notion that an information system adhering to this methodology will allow auditors to rely more heavily on the system as opposed to actually auditing the system itself.

Future research includes extending the proposed framework / approach to include details of the syntax to apply to business rule documentation, examining how auditors deal with analysis of the documents in audits of such systems and building full-scale systems to show that they are scalable and robust.

References

Brownston, L., Farrell, R., Kant, E., Martin, N., Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Addison-Wesley, Reading, Mass.

Chen, P.P., "The Entity-Relationship Model: Toward a Unified View of Data." *ACM Transactions on Database Systems* (January 1976): 9-36.

Dayal, U., and Widom, J., Active Database Systems, in ACM SIGMOD International Conference on Management of Data (tutorial), San Diego, Ca., June 1992.

Dayal, U., Hanson, E., and Widom, J., Active Database Systems, in Modern Database Systems, ed. Won Kim, Addison Wesley Publishing Co., ACM Press, New York, New York, 1995, pp. 434-456.

Dinn, A., Paton, N., and M. H. Williams, "Active Rule Analysis and Optimisation in the Rock & Roll Deductive Object-Oriented Databaset." Information Systems (Volume 24 Number 4 1999) page 327-353.

Kroenke, D., *Database Processing Fundamentals, Design and Implementation*, Prentice Hall, Upper Saddle River NJ, 2002.

McCarthy, W. E. The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. The Accounting Review vol. 57 No. 3 July 1982 pp. 554-578.

Von Halle, B. Lessons to Learn from Tee-ball. Database Programming & Design, December 1994.

Von Halle, Barbara, Explaining The Business Connection. Database Programming & Design, December, 1996.

Von Halle, Barbara, The Art Of Letting Go. Database Programming & Design, February, 1997.

Widom, J., Research Issues in Active Database Systems: Report from the Closing Panel at Ride-Ads '94, in ACM SIGMOD Record, 23(3):41-43, September 1994.

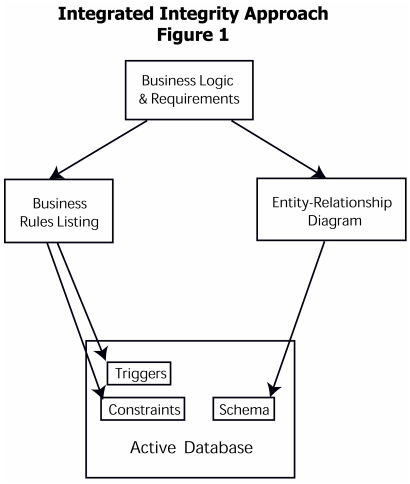

Figure 2

```
CREATE TABLE Customers
(
     CustomerID      INT        NOT NULL        PRIMARY KEY,
     FirstName       CHAR(20)   NULL,
     LastName        CHAR(20)   NULL,
     Address         CHAR(25)   NULL,
     City            CHAR(25)   NULL,
     State           CHAR(2)    NULL,
     ZipCode         CHAR(10)   NULL,
     Country         CHAR(20)   NULL,
     Phone           CHAR(20)   NULL,
     Fax             CHAR(15)   NULL,
     Email           CHAR(20)   NULL,
     ConfirmedCorrectness CHAR(1) NULL ,
     CreditLimit     DECIMAL(23,13) NULL DEFAULT (0),
     CurrentBalance DECIMAL(23,13) NULL
)

CREATE TABLE Orders (
     OrderID         INT        NOT NULL  PRIMARY KEY,
     CustomerID      INT        NOT NULL ,
     DateOrdered     DATETIME   NULL ,
     EstDeliveryDate DATETIME   NULL ,
     TotalOrder      DECIMAL(23,13) NULL ,
     PaymentMethod   CHAR(10)   NULL  DEFAULT ('Credit') ,
     Posted          CHAR(1)    NULL  DEFAULT ('n'),
     PaidInFull      CHAR(1)    NULL DEFAULT ('n'),
     FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)
)
```

This SQL declaration assures that if no credit limit is input for a given instance of a customer that the customer's initial credit limit will be zero and assures that a proper decimal credit limit will be entered.

This SQL declaration ensures that any Customer involved in an order must be a customer that exists in the database