Continuous Auditing

Continuous Auditing of Database Applications: An Embedded Audit Module Approach[1]

S. Michael Groomer, Uday S. Murthy, [2]

Article information:

To cite this document: S. Michael Groomer, Uday S. Murthy, [2]. "Continuous Auditing of Database Applications: An Embedded Audit Module Approach[1]" *In* Continuous Auditing. Published online: 12 Mar 2018; 105-124.
Permanent link to this document:
https://doi.org/10.1108/978-1-78743-413-420181005

For Authors

About Emerald www.emeraldinsight.com

Continuous Auditing of Database Applications: An Embedded Audit Module Approach[1]

S. Michael Groomer and Uday S. Murthy[2]

Abstract

This paper demonstrates an approach to address the unique control and security concerns in database environments by using audit modules embedded into application programs. Embedded audit modules (EAM) are sections of code built into application programs that capture information of audit significance on a continuous basis. The implementation of EAMs is presented using INGRESS a relational database management system. An interface which enables the auditor to access audit-related information stored in the database is also presented. The use of EAMs as an audit tool for compliance and substantive testing is discussed. Advantages and disadvantages of employing EAMs in database environments and future directions in this line of research are discussed.

ADVANCES in computer technology over the past several years have made computer-based accounting systems increasingly complex. With the virtual elimination of the traditional audit trail in computerized systems [Weber, 1982), internal control and system security are critical concerns. With recent improvements in computer technology and reductions in hardware costs, database management systems [DBMS] have become commonly used for business data processing. While there are unique control and security concerns relative to DBMS applications [Fernandez

[1]From *Journal of Information Systems 3*(2), 53–69. Reprinted by permission of American Accounting Association.

[2]S. Michael Groomer is Associate Professor of Accounting and Uday S. Murthy is Doctoral Candidate, both of the Department of Accounting, Indiana University, and Bloomington, Indiana 47405

Continuous Auditing: Theory and Application, 105–124

ISBN: 978-1-78743-414-1/doi:10.1108/978-1-78743-413-420181005

et al., 1981, p. 151), there is some evidence that auditors do not sufficiently adjust their audit procedures in the environment of DBMS [Roberts, 1980].

A variety of computer audit techniques have been discussed in the literature [Cash et al., 1977]. Though auditors have developed generalized audit software [GAS] to obtain audit evidence in advanced computer systems, such software packages are generally incompatible with the complex file structures of database systems [Cash et al., 1977, p. 824; Fernandez et al., 1981, p. 164]. Access to client database systems using GAS is most often done through intermediate sequential files extracted from the database. Embedded audit modules [EAM] are an example of concurrent auditing techniques [CAT] which continuously monitor transaction processing. Weber [1982, p. 475] highlights the increased need for CATs in the environment of database systems due to the integration of sub-systems and the sharing of data. Further, EAMs are perceived by auditors to be very efficient methods of auditing advanced computer-based systems [Tobison & Davis, 1981; Garsombke & Tabor, 1986].

The purpose of this paper is to describe an approach to the continuous auditing of database-driven accounting applications using EAMs. The objectives of this paper are to discuss (1) a selected number of unique control and security issues related to DBMS-driven accounting systems, (2) the use of a relational DBMS to construct and implement EAMs in a sales application, and (3) the utilization of EAMs as an audit tool. The significance of this paper lies in the demonstration of how audit modules embedded in a DBMS-driven application can address the unique control and security aspects of database environments. In addition, we demonstrate how the DBMS might enable the auditor to access audit-related information collected by the EAMs. In this manner, the illustration presented in this paper would assist auditors contemplating the use of EAMs in DBMS environments.

Need for EAMS in DBMS Environments

The unique control and security concerns in database environments can be addressed in two ways. One approach, which we present in this paper, is to use audit modules embedded into database-driven application programs. The advantage of this approach is that it can be employed regardless of the security and integrity features present in the particular DBMS being audited (or lack thereof). Another approach would be to utilize the security and integrity features of the DBMS software itself, if they are present and if they address the auditor's concerns. These features, which are programmed using the DBMS's data description language, might (1) prevent unauthorized accesses to database objects, (2) prevent unauthorized or erroneous updates to database objects, and (3) disallow erroneous transactions from being entered. The advantage of this approach is that controls need to be programmed only once-at the DBMS level. In contrast, the EAM approach requires controls to be programmed for each DBMS-driven application.

The extent to which present-day DBMS software contains the aforementioned security and integrity features is a question that is beyond the scope of this paper. When confronted with a particular DBMS environment, the auditor must determine the extent to which the built-in security and integrity features would address audit concerns. Since transaction processing in database environments invariably utilizes application programs, audit concerns in such environments can be effectively addressed by embedding audit modules into the application programs.

The next section of this paper discusses prior research, after which we discuss control and security concerns unique to database applications. We then demonstrate the implementation of EAMs in a relational DBMS. Thereafter, the use of EAMs as an audit tool is discussed. The advantages and drawbacks of using EAMs in database environments are then discussed. We conclude by discussing future directions in this line of research.

Prior Research

In a mail survey of 45 internal and 15 external auditors, Tobison and Davis [1981] addressed the actual use and perceived utility of several different electronic data processing [EDP] auditing techniques. Auditors were familiar with the use of audit modules and perceived them to be an effective technique, but only eight of them had used the technique in the past three years[3]. Auditors were most familiar with GAS.

Reeve [1984] surveyed the offices of Chartered Accounting Firms in Australia. The offices surveyed included all of the capital city offices of the Big-Eight accounting firms and a random selection of other offices of each of these firms. Sixty-six usable responses were returned. Reeve's findings support those of Tobison and Davis in that EAMs for both large and small systems were not found to be in wide usage. However, his survey results did indicate that the respondents expected to see increased usage of EAMs particularly for large scale systems.

In a field study of 245 internal EDP auditors, Moreish [1987] investigated the factors affecting the use of CATs. The internal auditors' involvement in the development of new systems, the maturity level of the EDP audit function, and the complexity of computerized systems were significant factors affecting the use of CATs. A specific advantage of using CATs cited was the ability to perform continuous monitoring of transactions in events-driven systems. The results of this study support the contention that EAMs are efficient methods of continuously auditing advanced computer-based systems.

Along these lines, Roberts [1980] investigated accounting control guide lines used by auditors to audit advanced computer systems. Internal control questionnaires from five Big-Eight firms were examined, and it was found that they were grossly

[3]In another mail survey by Garsombke and Tabor [1986] auditors rated EAMs as being very effective but were not very familiar with the technique.

inadequate for database systems (none of the 515 questions in the five different questionnaires specifically addressed DBMS controls such as data-base access controls). This suggests that auditors are not attuned to the idiosyncratic control and security concerns of database systems. However, it is conceivable that auditors have revised their control questionnaires and have become more aware of the control and security issues relative to DBMS in the years since the Roberts study.

Hansen and Messier [1984] have proposed a relational database management approach to providing the EDP auditor with a computer-based decision aid. Application controls installed in a database, the locations of these controls, the vulnerabilities resulting from their absence (risks) and possible exposures from those risks, are viewed as entities with certain attributes. The authors develop a conceptual schema of these entities which is mapped onto the database. The results of audit testing using a CAT (such as the Integrated Test Facility) are "captured" in a relation (the authors do not elaborate on this point) The auditor can then query the database by performing a series of relational algebraic operations (select, project and join) on the set of relations to answer queries such as-what are the risk exposures resulting from a specific failure to prepare a receiving report for incoming materials?

The approach used by Hansen and Messier is insightful. and provides a way to access data about transaction errors, once they have been captured. However, the authors do not present an approach to actually capturing transaction errors. The approach described in this paper uses EAMs in the transaction processing programs to capture and store information about errors as they occur in the database environments. Additionally, an interface is provided such that the auditors can access the stored information about actual errors and control violations. Gal and McCarthy [1985] demonstrate how integrity constraints can be implemented into a relational DBMS (Query-By-Example) consistent with traditional accounting concepts that govern segregation of duties. Integrity constraints are implemented by means of "views" to the database[4]. The method requires specification of views consistent with particular job functions and the assertion of authority constraints in these views.

Transactions that violate defined constraints are either disallowed with an error message or allowed with a warning message to the user. However, the fact of the error is not stored in the database for subsequent access by the auditor. The Gal and McCarthy [1985] approach utilizes functions within the data description language (DDL) of the DBMS software. Using this DDL approach, it is not possible to capture information about transaction errors and access violations for subsequent access by the auditor. Our approach, described below, utilizes audit modules embedded into the application programs (rather than using the DDL) and stores information of audit significance in a separate table in the database.

[4]McFadden and Hoffer [1985, p. 482-41:3] discuss how views can be defined in a database in such a way that users are allowed access to only a subset of data through that particular view.

Database Control and Security, and EAMS

This section discusses the unique control and security considerations relative to database systems from an audit standpoint. We also discuss how audit modules embedded into transaction processing programs that interface with the DBMS can address these unique control and security concerns.

Database Control and Security

Database accounting applications are characterized by increasing integration of functions and data that are typically separated in a manual or file-oriented computer system [Fernandez et al., 1981, p. 151]. The effect of such integration is that redundancy and independent sources of comparison are lost. The traditional audit trail thus becomes increasingly obscured. Integration of data further implies that data items that are common to multiple applications are stored only once and are "shared" by different applications. Sharing of data elements causes the different application subsystems to be tightly coupled, each relying heavily on the correct functioning of the other [Weber, 1982, p. 475].

Sharing of data makes it increasingly important to identify the "owner" of the data. Only the owner of the data should be authorized to update data (i.e., append, delete and modify) while others may be allowed read access. Further, there may be some data that are so sensitive that even read access must be denied to users. The need to ascertain that only authorized transactions are processed through the system is thus heightened. Further, the sheer size of many databases makes it difficult to scan them for errors [Fernandez et al., 1981, p. 151].

The idiosyncratic nature of DBMS applications necessitates focusing on access violations (i.e., access to the database) and control violations (in light of the concentrated nature of processing and the immense size of databases). Access violations comprise updates or retrievals of data by employees not authorized to do so. Control violations consist of transactions that are erroneous from an audit perspective (e.g., paying a nonexistent supplier). The increased concentration and integration of the transaction processing functions in DBMS environments necessitates intensified concern that proper authorization exists for all transactions that are processed by the DBMS. Furthermore, the auditor needs assurance that no violations occur with regard to management's approval of accesses to database entities.

Control Approaches in DBMS Environments

Establishing Controls Using DBMS Facilities

One approach to establishing controls in database environments is to use functions within the data description language (DDL) of the DBMS. Specifically, "permissions" can be granted on tables in the database to certain users, restricting the fields and records they may access and the times during which such access may be

performed. Further, "integrities" can be specified to ensure that, for example, all sales amounts are positive. Unfortunately, most existing database definition and manipulation languages, including SQL (Structured Query Language) [Astrahan et al., 1976], do not provide such facilities.

INGRES[5], 4 for example, allows the specification of permits (DEFINE PERMIT) and integrities (DEFINE INTEGRITY) at the individual table level. If a user attempts to perform functions that violate the permissions granted to him/her, an error message appears and the transaction is aborted. Similarly, if integrity is violated during run time, an error message is displayed and the transaction is not processed. However, no information about such violations is captured. Ideally, every exception should trigger functions that record information about the violation in separate tables in the database. While the DDL in INGRES provides the system developer with the ability to define permits and integrities, it does not satisfy audit requirements since information about exceptions cannot be captured as they occur. An alternative approach, discussed below, is to embed audit modules into the application programs to capture information about these control and access violations as they occur.

The advantage of using the DDL to satisfy control and security concerns is that it promotes data independence. Permits and integrities defined at the level of individual tables do not have to be changed if modifications are made to application programs accessing the same tables. However, as noted earlier, the disadvantage of using the DDL in INGRES is that information about the violations cannot be captured. With the move towards standardization of SQL as the predominant database definition and manipulation language, it is noteworthy that it lacks features such as permits and integrities. Such features, together with the ability to capture information about violations as they occur, should be investigated for possible inclusion in future versions of SQL.

Audit Modules Embedded Into DBMS Application Programs

An alternative approach to using the DDL to specify permits and integrities is to embed modules into the application programs used to process transactions in database environments. Transaction processing in database systems is accomplished by means of application programs that interface with the DBMS software to access the relevant database entities. The application programs themselves are written in widely used third-generation languages (e.g., COBOL and FORTRAN). To interface with the DBMS, these application programs have statements that "call" the database and manipulate data elements in the syntax of the particular implementation's database manipulation language. The relationship between the application program, the DBMS software, and the database itself is depicted in Figure 1.

[5]INGRES, a full-featured relational DBMS, is a product of Relational Technology Inc. [1986]. The version of INGRES used in this study is a full-featured variant implemented on a DEC VAX 11/785, running under the VMS operating system.

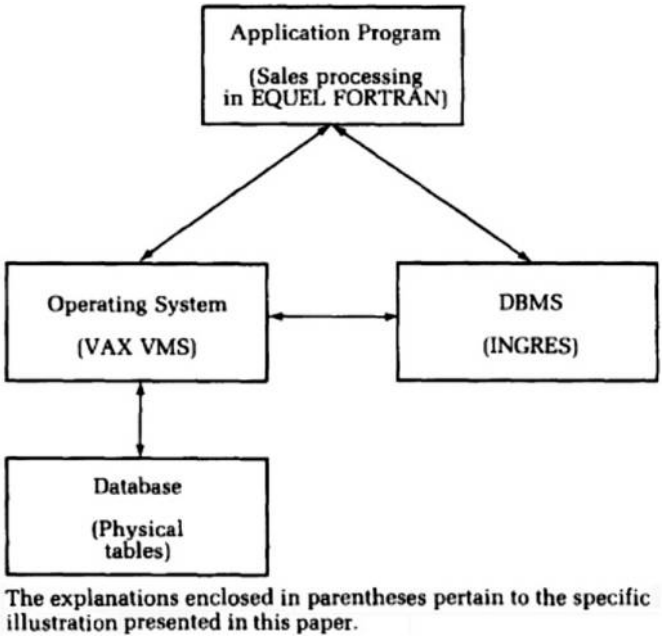

Figure 1: Application Programs in DBMS Environments.

We propose that EAMs incorporated into application programs can address the audit objectives related to the special control and security concerns outlined earlier. Essentially, EAMs are modules (code) built into application programs that are designed to capture audit-related information on an ongoing basis. This automated capture of audit-related information constitutes a substantive test of all transactions processed during the year. From a programming standpoint, audit modules are subroutines that are invoked whenever certain conditions of audit significance are met. The function of the subroutines is to record facts about erroneous transactions in the database so that the information can subsequently be accessed by the auditor, either through programs developed to generate specially designed reports or by using the database query facility to generate the desired reports.

All routine transaction processing activity must necessarily use the application programs[6]. Properly designed EAMs will detect and capture information of audit significance as actual transactions are processed. Thus, EAMs constitute a monitoring system resident in the application supported by the DBMS that provides a "continuous audit" of a particular accounting application.

[6]We assume that the majority of routine transactions are processed using the application programs designed for that purpose. While transactions could certainly be processed by directly accessing the database using the query language, sound management practices would dictate that only personnel authorized by the database administrator could do so.

Auditing Database Applications Using EAMS

In this section, we first discuss alternative approaches to controlling transaction errors and access violations. We then demonstrate how audit modules implemented into a sales transaction processing program can detect and store information about control and access violations in a relational DBMS. An interface that allows the auditor to access information about control and access violations is also described.

Error Handling Approaches

There are two approaches to the handling of transactions that constitute control or access violations. The transactions in error could either be disallowed, or they could be allowed and the fact of the error stored in control tables. Gal and McCarthy [1985, p. 30] describe these two approaches as "pre-operative" and "post-operative" constraint checking sequences. The pre-operative approach detects errors in transactions and disallows them. The post-operative approach also detects the error, but the transaction is executed.

Pragmatically, the accounting system will have to be designed such that transaction processing proceeds expeditiously. For instance, rather than being unable to process a credit memorandum due to the absence of a particular manager, the transaction will most likely be processed and approved subsequently. However, certain errors will be significant enough to disallow the transaction from being executed. The nature of errors that would be sufficient cause to disallow transactions, and errors that can be "conditionally" accepted, is a design consideration that must be worked out with the client. In our example, we assume that erroneous transactions are to be allowed, with the fact of the error stored in a control table.

Access violations, on the other hand, are of such significance that they must not only be detected but information regarding them should be captured. Illegal access to the database must be prevented, with particulars of the attempted access stored for review by the auditor. Wong [1985, p. 294] discusses the importance of monitoring and regularly reviewing access violations. Straub [1986] discusses the effectiveness of administrative methods to deter such violations.

Implementation of EAMS

The database and an example sales transaction processing module are developed in INGRES. The system designed accepts transactions "on-line," i.e., sales invoices are directly entered into the database using the sales application program.

Audit modules are embedded into this program. Figure 2 shows a portion of the program where the EAMs are "called," and also the EAM subroutine that captures credit approval and credit limit violations. The program is written in EQUEL

(Embedded QUEry Language) FORTRAN. EQUEL programs allow the programmer to embed a fourth generation database query language (QUEL) into a third generation language (FORTRAN). QUEL and the popular SQL differ primarily in their syntax: their functions are very similar.

```
THE FOLLOWING IS A PART OF THE SALES TRANSACTION PROGRAM WHERE THE
"CREDVIOL" SUBROUTINE IS CALLED.
C------
C   Take user to the 'amount' field in the 'sales' form
C------
##          activate field 'amount':
##          {
C------
C   Get amount of the sale
C------
##          getform (f_amount = amount)
C------
C   If amount > 10000 then activate the 'credviol' subroutine
C------
            if (f_amount .GT. 10000) then
              call credviol
            endif
C------
C   If amount > 5000 then activate the amtviol subroutine
C------
            if (f_amount .GT. 5000) then
              call amtviol
            endif
C------
C   Take user to next field in the 'sales' form
C------
##          resume field invoice_no
##          }

THE FOLLOWING SUBROUTINE CHECKS WHETHER CREDIT HAS BEEN APPROVED
FOR SALES ABOVE $10000, AND WHETHER THE AMOUNT OF CREDIT APPROVAL
HAS BEEN EXCEEDED.
            subroutine credviol
C------
C   Define variables that are common between this
C   subroutine & main program
C------
            common f_pers_id, f_invoice_no, f_amount, f_cred_auth,
                   f_customer
##          declare
C------
C   Variable definitions for this subroutine
C------
##            character*8     f_pers_id
##            character*4     f_cred_auth
##            real*4          f_amount, f_authamt
##            integer*4       f_invoice_no
##            character*20    f_customer
C------
C   Initialize the logical variable found1 to 'false'
C------
            found1 = .FALSE.
C------
C   Access the 'credauth' table & retrieve credit
C   authorization amount for this customer, for the
```

Figure 2: Part of the Sales Transaction Program Where the "Credviol" Subroutine is Called.

```
C   particular 'cred_auth' number
C------
##          retrieve (f_authamt = cr.amount) where f_customer =
##              cr.customer and f_cred_auth = cr.cred_auth
C------
C   If the retrieve was successful, indicating that there
C   exists an entry in the 'credauth' table for this customer
C   and for the particular credit authorization number,
C   the statement between the next two curly braces is
C   executed and the logical 'found1' evaluates to 'true'.
C------
##          {
             found1 = .TRUE.
##          }
C------
C   Now check if found1 is 'false'. If found1 is false, then
C   it means that there is no entry in the 'credauth' table
C   for this customer. This is a control violation which is
C   stored in the 'violations' table.
C------
            if (found1 .EQ. .FALSE. ) then
##            append to violations (invoice_no = f_invoice no,
##              personal_id = f_pers_id, viol_date = 'today',
##              viol_desc = 'NO APPROVED CREDIT', amount = f_amount,
##              viol_type = 'C')
            endif
C------
C   Now check to see if the authorized credit amount has
C   been exceeded. If so, record particulars of the violation
C   in the 'violations' table.
C------
            if ((f_amount .GT. f_authamt) .AND.
     1        (found1 .EQ. .TRUE.)) then
##          append to violations (invoice_no = f_invoice_no,
##              personal_id = f_pers_id, viol_date = 'today',
##              viol_desc = 'CREDIT APPROVAL EXCEEDED',
##              amount = f_amount, viol_type = 'C')
            endif
C------
C   Return control to the main program
C------
            return
            end
```

Figure 2: (*Continued*)

The extent of familiarity with the programming environment required to implement EAMs in database environments is significant. Computer audit specialists[7] must be conversant with both the host language in which the application program is written (FORTRAN in our example) and the query language of the particular DBMS (QUEL in our example).

[7]We envision that the design and implementation of EAMs will primarily be performed by internal auditors with computer expertise or by computer audit specialists in the major accounting firms if external auditors participate. However, accessing information of audit significance captured by the EAMs could be done by personnel who may not necessarily be computer audit specialists, such as senior accountants.

The tables used in this example are shown in Figure 3. The 'employees' table shows the personal identification number of each employee, his or her department, the grade in the department, and the employee's password. The 'authmatrix' table contains the employee personal identification number, the table (i.e., sales) that the employee may access, and the type of access permissible. The 'credauth' table shows particulars of customers to whom credit has been extended. A list of authorized customers is shown in the 'custlist' table. Sales invoices are stored in the

EMPLOYEES table

personal_id	name	department	grade	password
JD222	JANE DOE	FINANCE	CLERK	JNXYZ
MB456	MARY BROWN	ACCOUNTING	MANAGER	MBPW
TM123	TOM JONES	SALES	MANAGER	TOMMY
JD111	JOHN DOE	MARKETING	CLERK	JOHNMKT
PC101	PHIL COLLINS	PURCHASING	MANAGER	PHILCOL
RD999	ROGER DALTREY	SALES	CLERK	ROGERD7

AUTHMATRIX table

personal_id	table	access
TM123	SALES	U
JD111	SALES	R
MB456	SALES	U
JD222	SALES	R
PC101	PURCHASES	U
RD999	SALES	U

CUSTLIST table

customer	add1	add2
XYZ CO.	123 NOWHERE ST.	ANYTOWN, USA
ABC CO.	300 WALNUT ST.	ANYTOWN, USA
BIGBUX CO.	1000 MONEY DR.	DOUGHTOWN, USA

CREDAUTH table

cdate	cred_auth	customer	amount	personal_id
24-sep-1987	C100	XYZ CO.	12000.000	TM123
24-sep-1987	C101	ABC CO.	20000.000	TM123

SALESINV table

invoice	invdate	personal	customer	cred_auth	item	amount
1001	26-sep-1987	RD999	ABC CO.		5 ENGINE PARTS	2450.000
1002	26-sep-1987	TM123	XYZ CO.	C100	10 SPARE PARTS	11000.000
1003	26-sep-1987	JD111	BIGBUX CO.		15 DRILLING BIT	14000.000
1004	26-sep-1987	TM123	DEF CO.		5 WIDGETS	2000.000
1005	26-sep-1987	TM123	ABC CO.	C101	20 ELECTRIC ENG	24500.000
1006	26-sep-1987	RD999	ABC CO.		12 CUTTING TOOL	65000.000

VIOLATIONS table

invoice	personal_id	viol_date	viol_type	viol_desc	amount
1003	TM123	26-sep-1987	C	NO APPROVED CREDIT	14000.000
1004	TM123	26-sep-1987	C	CUSTOMER NOT APPROVED	0.000
1005	TM123	26-sep-1987	C	CREDIT APPROVAL EXCEEDED	24500.000
1006	RD999	26-sep-1987	C	SALE NOT APPROVED BY MANAGER	6500.000
0	PC101	26-sep-1987	A	UNAUTHORIZED ACCESS TO SALES	0.000
1003	JD111	26-sep-1987	A	UNAUTHORIZED UPDATE	14000.000

Figure 3: Tables and their contents.

'salesinv' table. Control and access violations captured by the EAMs are stored in the 'violations' table as they occur.

Control Violations

For demonstration purposes, the following types of transaction errors are designed to be captured: (1) sales invoices above $10,000 for which pre-approved credit has not been obtained, (2) sales to customers who are not on the approved customer list (see the 'custlist' table in Figures 3), and (3) sales invoices above $5,000 which have been prepared by someone other than the sales manager. Pre-approved credit is checked by reference to the 'credauth' table which indicates the customer name and the amount of credit approval. Of course, in an actual sales transaction processing system the audit modules would be designed to capture many more types of errors.

The moment these errors are detected by the audit modules (i.e., as the sales transactions are being processed), the following information about the errors is stored in the 'violations' table: (1) a transaction identifier (i.e., the invoice number), (2) the personal identification number of the employee who input the transaction, (3) the date of the error, (4) the fact that the error is a control violation ('C'), (5) the exact description of the violation, and (6) the dollar amount of the transaction. Additional particulars, such as the name and department of the employee, can be obtained by joining the 'violations' table with the 'employees' table (the 'personal_id' field is the basis for the join; an example is provided at the end of this section).

Access Violations

As stated earlier, one of the purposes of the EAMs in the database environment is to ensure that only the owner of data updates or adds data items, while non–owners may be allowed read access to the data. In our example, it is assumed that all employees above a certain grade in the sales department are authorized to update (modify or delete) or add sales records. For example, Tom Jones (personal id = TM123), a manager in the sales department, is allowed update access to the sales table (see the 'authmatrix' table in Figure 3). Other users in the organization, e.g., accounts receivable, credit approval, marketing, etc., will be allowed read access only. In our example, John Doe (personal id = JD111) is allowed read access only to the sales table (see Figure 3). Note that Phil Collins (personal id = PC101) is allowed access to the purchases table but not to the sales table.

The audit modules are designed to capture the following information about unauthorized accesses to the sales table: (1) the personal identification number of the employee who accessed the table, (2) the date of the unauthorized access, (3) the fact that the error is an access violation ('A'), (4) the exact description of the violation, and (5) the dollar amount of the transaction. As with control violations,

particulars such as the name and department of the employee, can be obtained by joining the 'violations' table with the 'employees' table.

The basis for the detection of access violations is an 'authorization matrix' (see the 'authmatrix' table in Figure 3; 'R' stands for read-only access and 'U' for update access). Of course, appropriate protection will have to be provided for the authorization matrix itself to prevent illegal changes to it. When the sales application program is invoked, the employee is asked to enter his or her personal identification number and pass word. This is checked by referring to the 'employees' table where the personal identification number, name, and password of all employees is stored.

Rather than granting access to particular employees, a more robust approach is to grant access to organizational positions. The 'authmatrix' table can specify table accesses based on the 'grade' field rather than the 'personal_id' field. Thus, when employees change positions or leave the organization, changes need be made only to the 'employees' table and not to the 'authmatrix' table as well.

Auditor Interface

Audit relevant information is stored in a separate violations table. Note that one of the attributes of this table is "viol_type" which identifies whether the violation is a control violation or an access violation (see the 'violations' table in Figure 3). The table also identifies the employee who originated the transaction (i.e., through the 'personal_id' field) and describes the violation ('viol_desc'). The amount of the transaction is also stored. As with the 'authmatrix' table, appropriate protections will have to be specified so that modifications and deletions to the 'violations' table are prevented.

The auditor can access information stored in the 'violations' table by invoking report generation programs at the operating system level. These programs are written using the "Report Writer" capabilities of INGRES. At periodic intervals the auditor can run these report programs to determine what transaction errors (viol_type = 'C') and access violations (viol_type = 'A') have occurred. Note that the dollar amount of the error has been captured and stored in each case. In keeping with the auditor's materiality threshold, the report-writer programs are designed so that only errors that meet or exceed the materiality threshold are reported. The materiality level parameter is entered by the auditor at the time the report-writer program is invoked. Figure 4 shows sample outputs from running the INGRES report-writer programs.

The report-writer programs could be used by both the internal and the external auditor. In fact, the internal auditor may want to investigate errors below the external auditor's materiality threshold since they may be indicative of user difficulties with the system. For instance, the errors may stern from lack of personnel training, ineffective supervision, or poor documentation.

If the internal and external auditors are familiar with a fourth-generation database query language (QUEL in the case of INGRES), they could directly access the 'violations' table and generate reports based on ad hoc queries. For example, to

REPORT OF CONTROL VIOLATIONS

REPORT DATE: 9/26/87

Materiality level: 5000

Viol. Date	PID	Inv. No.	Description	Amount
9/26/87	TM123	1003	NO APPROVED CREDIT	$14,000.00
9/26/87	TM123	1005	CREDIT APPROVAL EXCEEDED	$24,500.00
9/26/87	RD999	1006	SALE NOT APPROVED BY MGR	$6,500.00
TOTAL OF ALL VIOLATIONS				$45,000.00

REPORT OF CONTROL VIOLATIONS

REPORT DATE: 9/26/87

Materiality level: 0

Viol_Date	PID	Inv. No.	Description	Amount
9/26/87	JD111	1003	UNAUTHORIZED UPDATE	$14,000.00
9/26/87	PC101	0	UNAUTHORIZED ACCESS TO SALES	
TOTAL OF ALL VIOLATIONS				$14,000.00

Figure 4: Report of Control Violations.

RESULT OF AD-HOC QUERY IN QUEL

viol_desc	amount	name	department
NO APPROVED CREDIT	14000.000	TOM JONES	SALES
CUSTOMER NOT APPROVED	0.000	TOM JONES	SALES
CREDIT APPROVAL EXCEEDED	24500.000	TOM JONES	SALES
SALE NOT APPROVED BY MANAGER	6500.000	ROGER DALTREY	SALES

Figure 5: Result of AD-HOC Query in Quel.

determine the violation description, the amount and the names and departments of employee(s) responsible for all control violations on a particular day (9/26/87), the auditor would enter the following query in QUEL:

```
range of e is employees
range of v is violations
retrieve (v.viol_desc, v.arnount,e.narne,e.departrnent) where:
v.vioLtype = "C"
and v.viol date = "9/26/87"
and v.personaLid = e.personaLid
```

The result of entering the above query is shown in Figure 5. The query operates by joining the employees table with the violations table on the common field,

'personal_id.' Note that only selected fields from the 'violations' and 'employees' table are extracted in this ad-hoc query.

The process of implementing EAMs into application programs may be summarized as follows: Step 1-Deterrnine application to be audited; Step 2-Deterrnine audit objectives relative to the application; Step 3-Identify triggers in the application program which are to be considered as control and access violations; Step 4-Determine audit relevant information to be captured when the trigger conditions occur; Step 5-Write audit modules (e.g., subroutines in FORTRAN or procedures in PASCAL) to capture audit relevant information in separate database entities; Step 6-Program the trigger conditions that invoke the audit modules at various points in the application program; Step 7-Create database entities in which the audit modules would store audit relevant information (e.g., tables in a relational DBMS); Step a-Recompile and test the application program to ensure that the audit modules are operating as desired; Step 9-Place the new version of the program in the production library. These steps are depicted in Figure 6.

In the next section, we discuss how EAMs can be used as an audit tool for compliance testing or for substantive testing.

Use of EAMS as an Audit Tool

EAMs in database applications can operate both as compliance testing and substantive testing tools. EAMs can potentially capture information about all transaction errors and control violations. However, whether EAMs are used for compliance testing or for substantive testing or for both purposes (i.e., as a dual-purpose test) depends on whether they are operative continuously or only periodically.

If EAMs are operative continually, throughout the accounting period, they constitute a very comprehensive compliance and substantive testing audit tool. The auditor would have information about the operation of controls (i.e., EAMs as compliance testing tools), as well as information about actual transaction errors (i.e., EAMs as substantive testing tools), as well as information about actual transaction errors (i.e., EAMs as substantive testing tools). Thus, EAMs can operate to facilitate dual-purpose testing. The extent of traditional year-end substantive tests of transactions and balances can be virtually eliminated-the auditor needs only to access the control tables for information about control violations and transaction errors.

However, due to operational considerations (discussed in the following section), EAMs may be operative only during a part of the accounting period. In this event, the auditor may choose to "turn on" the EAMs during the interim audit. Information about control and access violations captured by the EAMs during the interim audit serves as information from a compliance testing perspective. After evaluating the audit evidence collected by the EAMs during the interim audit, the auditor may decide to curtail the extent of substantive tests of transactions and balances at year end. Furthermore, since EAMs can operate as a substantive testing tool, the auditor could simply activate the EAMs during the sensitive months

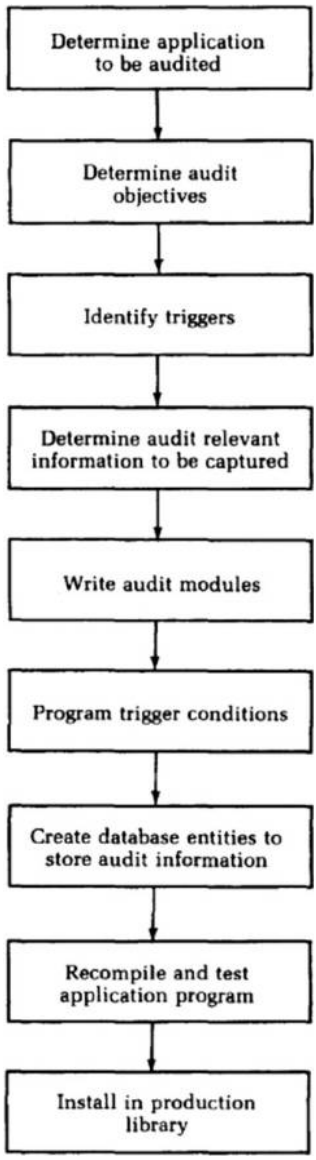

Figure 6: The EAM Implementation Process.

around year-end. In this manner, the auditor can perform an efficient substantive test of the client's transactions and balances at year-end.

The versatility of EAMs is thus apparent: they can operate in a dual purpose mode if operative continually throughout the audit period, or they can be employed to compliance test controls during the interim audit and to perform substantive tests of transactions and balances at year-end. Which option is chosen depends in large

part on the evaluation of the overall client environment and the audit objectives taken as a whole. For instance, if frequent control violations and transaction errors are found during the interim audit, it may be deemed necessary to have the EAMs operational throughout the remainder of the year. On the other hand, in a stable, well controlled client environment, the EAMs may be employed only intermittently.

In the following section we discuss the various operational considerations in using EAMs and also point out the advantages and disadvantages of the EAM approach.

Advantages and Drawbacks of EAMS

This section discusses some of the operational considerations to be evaluated when implementing EAMs into a database-oriented application environment. We also highlight the advantages and drawbacks of employing EAMs in database environments.

Operational Considerations

From the client's point of view, the use of EAMs in application programs may cause an undesirable impact on system performance. The overhead attributable to EAMs may be substantial, especially if the nature and extent of control procedures to be examined is extensive. Therefore, due to hardware and software performance considerations, the system of EAMs could be designed so that they can be "turned on and off" at the auditor's discretion. A separate 'auditor' table can be constructed in which the auditor can specify dates when the audit modules are to be operational. When the sales application program is invoked, the 'auditor' table can be accessed to determine whether the auditor has specified that the modules are to be operationalized during that particular period.

Information collected in the 'violations' table may be fairly extensive and thereby costly to store in an on-line mode. However, even with existing computer technology, data collected by EAMs can be transferred periodically to inexpensive storage media. Technical improvements in hardware and software may very well facilitate the utilization of EAMs on a full time basis, even in large installations.

A critical concern is control over the EAMs themselves. In the auditor's review of general controls, special attention must be paid to control over program modifications and programmer access to application source codes. Additionally, periodic checks will have to be made to ensure that the modules are in fact operational. This can be accomplished by processing test transactions online to determine whether the modules capture information about control and access violations as they were designed.

Advantages of Using EAMS

The use of EAMs should be considered advantageous for a number of reasons. First, information about control violations and dollar errors is captured on a

continuous, real time basis. If EAMs are not present, the audit organization would have no choice but to employ traditional ex-post auditing approaches. Traditional sampling approaches require the auditor to infer the quality of accounting outputs based on results of applying audit procedures to the sample. Second, using EAMs the audit organization is not confined to sampling processes at the traditional interim or year-end periods. Since properly designed EAMs should capture all transaction errors, substantive testing at year end should be virtually eliminated when compared to traditional sampling-approaches. Third, EAMs provide a superior method of ensuring that material errors or control violations are trapped. Even if EAMs are operative only intermittently, sampling risk is reduced, since the auditor has knowledge of all errors in the periods sampled. Fourth, since the audit organization can capture control violations and dollar errors at will, this approach would seemingly reduce the extent of compliance testing compared to approaches where EAMs are not used. Fifth, where EAMs are to be used only intermittently, their use provides a "surprise" test capability, since the application personnel should not be aware that the auditor has activated the EAMs.

Drawbacks

There are a number of drawbacks to the use of EAMs. First, the computer audit specialist must not only be knowledge able about DBMS, but also must be knowledgeable of the application environment, particularly with respect to the control environment, host and query languages. With the increasing complexity of computer-based environments, we believe that auditor comprehension of the mechanics of the client's computer-driven systems is unavoidable. Second, client cooperation to allow the implementation of EAMs is necessary. The auditor must be able to demonstrate and convince the client of the merit of such an undertaking. We suggest that the primary point of interest is a reduction of audit fees over the long run, since the benefits of EAMs will accrue over several accounting periods. Moreover, the auditor should indicate that improvement of client systems is also a by-product. However, the "selling" of these concerns to the client may not be an easy task.

The third drawback is that EAMs are not viable where the target client system is unstable. If the client system is in a constant state of flux, the audit modules, if implemented, will likely have to undergo modification at frequent intervals thus raising the cost of the audit modules approach. Regardless, the use of these modules requires an on-going commitment of personnel and funds by the client and the auditor. We believe that the most effective implementation of EAMs can be accomplished where auditors are involved in the systems development process. Auditor involvement will likely result in not only an effectively designed, implemented and controlled system, but a system in which consideration was given to the use of EAMs at the initial stages of design. This should result in well designed and cost effective EAMs.

Conclusion

The approach described above, using EAMs in database environments, is important in that it demonstrates the feasibility of addressing the idiosyncratic control and security concerns relative to DBMS-driven applications. In addition, the approach illustrates how audits of such applications can be performed in an efficient manner. A superior approach would be to use the DDL to specify permits and integrity constraints, if information on violations could be captured as they occur. Future developments in database definition and manipulation languages may incorporate such features, which would obviate the need to embed audit modules in application programs.

Further research should investigate whether auditors are more attuned now to the special control and security problems present in database environments. The cost-benefit aspect of using EAMs should also be addressed. Specifically, the impact on system performance of incorporating audit modules should be investigated. There are suggestions in the literature that auditors should be more involved in the design of accounting systems [e.g., Grabski, 1986]. As indicated in the previous section, we suggest that auditors should propose the implementation of EAMs in application programs to capture and store information of audit significance on a continuous basis. Further research could investigate whether the utilization of more efficient concurrent auditing techniques (i.e., EAMs) leads to decreased audit risk in the environment of complex database accounting systems.

References

Astrahan, M. M., M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones,]. W. Mehl, G. R. Putzolo, I. L. Traiger, B. W. Wade, and V. Watson, "System-R: Relational Approach to Database Management," *ACM Transactions on Database Systems* (June 1976). pp. 97–137.

Biggs, S. F., W. F. Messier, Jr., and J. V. Hansen, "A Descriptive Analysis of Computer Audit Specialists' Decision-Making Behavior in Advanced Computer Environments," *Auditing: A Journal of Practice & Theory* (Spring 1987), pp. 1–21.

Cash, J. I., Jr., A. D., Bailey, Jr., and A. B. Whinston, "A Survey of Techniques foR Auditing EDP-Based Accounting Information Systems," *The Accounting Review* (October 1977), pp. 813–832.

Fernandez, E. B., R. C. Summers, and C. Wood, *Database Security and Integrity* (Reading, MA: Addison-Wesley Publishing Company, 1981).

Gal, G., and W. E. McCarthy, "Specification of Internal Controls in a Database Environment," *Computers and Security* (March 1985). pp. 23–32.

Garsombke, P. H., and R. H. Tabor, "Factors Explaining the Use of EDP AudiT Techniques," *The Journal of Information Systems* (Fall 1986), pp. 48–66.

Grabski, S., "Auditor Participation in Accounting Systems Design: Past Involvement and Future Challenges," *The Journal of Information Systems* (Fall 1986), pp. 3–23.

Hansen, J. V., and W. F. Messier, Jr., "A Relational Approach to Decision Support for EDP Auditing," *Communications of the ACM* (November 1984), pp. 1129–1133.

McFadden, F. R., and J. A. Hoffer, *Data Base Management* (Menlo Park, CA: The Benjamins/Cummings Publishing Company, 1985).

Mohrweis, L. C., "An Empirical Investigation of Factors Affecting the Use of Concurrent EDP Audit Techniques," Working Paper, Indiana University, September 1987.

Reeve, R. C., "Trends in the Use of EDP Audit Techniques." *The Australian Computer Journal* (May 1984), pp. 42–47.

Relational Technology, Inc., *INGRES Reference Manual* (Berkeley, CA: Relational Technology, Inc., 1986).

Roberts, M. B., "An Investigation of Guidelines for Review and Evaluation of Accounting Controls in Data Base Management Systems," Unpublished Ph.D. Dissertation, Georgia State University, 1980.

Straub, D., "Deterring Computer Abuse: The Effectiveness of Deterrent Countermeasures in The Computer Security Environment," Unpublished Ph.D. dissertation, Indiana University, 1986.

Tobison, G. L., and G. B. Davis, "Actual Use and Perceived Utility of EDP Auditing Techniques," *The EDP Auditor* (Spring 1981), pp. 1–22.

Weber, R., *EDP Auditing: Conceptual Foundations and Practice* (New York, NY: McGraw-Hill Book Company, 1982).

Wong, K. "Computer Crime-Risk Management and Computer Security," *Computers and Security* (December 1985). pp. 287–295.