Specification of Internal Accounting Controls in a Database Environment *

Graham Gal
Dept. of Accounting, University of Massachusetts, Amherst, Massachusetts 01003, USA

and

William E. McCarthy
Dept. of Accounting, Michigan State University, East Lansing, Michigan 48824, USA

This paper examines the problem of specifying database security controls in a manner such that the resulting segmentation of data and the patterns of access rights are consistent with traditional accounting concepts that govern segregation of duties. The mechanism we use for implementation of these controls in a relational accounting system is that of a "view" implemented on the Query-by-Example database management system. A number of examples are presented in detail and some further aspects of security and integrity constraints are discussed.

Keywords: Internal accounting controls, separation of duties, integrity constraints, database views, relational database.

Graham Gal is an Assistant Professor of Accounting at the University of Massachusetts in Amherst. His Ph.D. is from Michigan State University in Accounting and Information Systems. His research interests include conceptual database models, conceptual control models, and artificial intelligence applications in accounting. He has published a number of articles that examine particular database implementations of accounting systems. His professional experience is in the area of computer systems design and implementation. He has worked as both an EDP supervisor and consultant to firms concerning the development of computer based accounting systems. He is currently a member of ACM and the American Accounting Association and was recently selected as a Society of Information Management Doctoral Fellow.

* This project was funded by a grant from the Peat, Marwick, Mitchell Foundation through its Research Opportunities in Auditing program. The views expressed to not necessarily reflect the views of the Peat, Marwick, Mitchell Foundation. A preliminary version of a portion of the work was presented at the 1982 Midwest meeting of the American Institute for Decision Sciences.

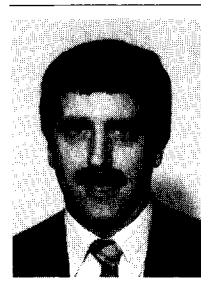
North-Holland
Computers & Security 4 (1985) 23-32

1. Introduction

Information systems can be characterized along many different dimensions. One such dimension pertains to the access rules by which users can obtain data. At one end of this dimension are those systems that have open access and at the other are those systems with closed access.

An open system is one in which the data is essentially open or available to all user groups. Data which is not to be used by certain users is then locked to restrict access. A library would be a good example of an information system with open access rules, because all information is generally available with the possible exception of overdue and fine information. In contrast to open systems are those that are characterized as having access rules which close the information.

A closed system is one in which all the information is locked or unavailable to users. Information that is required by a particular user or group of users is then unlocked or made available to these users. Military information systems are examples of this type of data availability. In these information systems the availability of data is dependent on the security clearance of a particular user, i.e. information is unlocked to users with appropriate levels of clearance. Business information systems can also be characterized as being closed with respect to access rules (for a good discussion of general security issues in a database management system (DBMS) see [4]).



William E. McCarthy is an Associate Professor of Accounting at Michigan State University. Professor McCarthy received his AB degree in Economics from Boston College and his MBA Southern Illinois University. His Ph.D. in Accounting was received from the University of Massachusetts in 1978. Professor McCarthy has published a number of database papers in both the accounting and the computer science literature, and he is currently doing research in areas such as data modeling, computer auditing, and expert system construction. He is presently the principal investigator on research grants received from the Peat, Marwick, Mitchell Foundation and the Touche Ross Foundation.

Business organizations need to process large amounts of data in order to make decisions that are pertinent to the success of the business and to satisfy various reporting requirements. It is also important, however, for companies to implement proper internal controls over the use and input of the data. In most companies, the unlocking, or granting of access to subsets of the corporate data is done according to the requirements of separation of duties, i.e. the access is based on data subsets which are related to job functions within a business organization [2].

Under traditional manual types of business data processing systems, the internal control over accounting data is effected by exercising physical control over the source documents, journals, ledgers and files. That is, the data is kept under lock and key and is made available to those users whose job function specifically requires certain pieces of data. As business organizations become larger and more complex, the data is stored less in journals and ledgers and more in computer readable storage media [tapes and disc]. Further, in a database environment, the data is centralized in corporate data pools with shared access among large groups of heterogeneous users through the facilities of a database management system (DBMS). In such an environment, the overall model of the firm's data is called a *conceptual schema*. The conceptual schema is then partitioned into a set of *logical views* (or external schemas) which would be consistent with the subset of the corporate information required for a particular job function or decision setting.

In order for this corporate data pool to be useful for accountants, especially as they fulfill external reporting requirements, it is necessary that proper internal controls be exerted over the data. While it is true that certain controls are of interest to many groups of users, accountants as a profession are interested in a particular group of controls that affect the accounting data. When auditors examine an information system that produces the data for the external reports, they require not only that data be restricted to the job function that needs the data but also that the assignment of the job functions be carried out with regard to proper separation of duties [2].

This paper addresses the formulation of views that would be consistent with particular job functions and the assertion of controls over these views. The views are formulated in an information system which has been specified and modeled in accordance with the principles of "events" accounting [9]. In an events system, transaction data are recorded and stored in disaggregate form, and the conceptual schema elements related to accounting consist of objects representing (1) economic events, (2) economic resources affected by the events, and (3) economic agents who participate in the events. This type of specification was selected, because it has been demonstrated that such a system can supply all of the traditional accounting requirements as well as filling the needs for accounting data by non-accountant users (see especially [5] and [10]).

The particular DBMS that will be used to demonstrate view construction and access control is a relational model of a small retail enterprise that has been implemented using the IBM software package Query-by-Example (QBE) [7,13]. In our particular implementation [6], economic resources, events and agents are modeled as sets of entities and relationships [8] and are then represented as *base tables* or relations in the system. Logical views are defined as *stored queries* or programs run against the appropriate tables. The access to both views and the underlying transaction data is controlled by declaring authorization constraints (also in the form of queries) that govern read, insert, update, and delete privileges within the system (for a general discussion of these facilities in DBMS see [3] and [12]). These authorization constraints are maintained in the data dictionary in the table called "AUTHORITY" which is checked before a query is performed.

The next section of the paper demonstrates the construction of views that are consistent with separation of duties and the establishment of controls over these views in accordance with the job functions. In the section following these examples, some of the limitations of the QBE implementation are examined in relation to the requirements of internal controls.

2. Formulation of Views and Control of Their Use

2.1. Accounts Receivable

Most business organizations make sales to customers and then collect payments for these

sales. For internal control purposes, the initiation and processing of these transactions are normally delegated to different job functions or employees within the firm. These functions are separated so as to ensure that corporate assets will not be misappropriated. Such a possibility would exist if the same employee not only monitored accounts receivable and was authorized to make adjustments to those accounts, but also was responsible for entering payments to those same accounts. In this case the clerk would be able to make an account adjustment and then change the recording of payments to conceal the misappropriation of funds.

These types of problems could be reduced, if not eliminated, in an events database through separation of the custodial, recording and operational duties and by establishing access rights in accordance with the "policy of least privilege" [4, p. 59]. This policy specifies that all users should be granted access to the smallest subset of the information pool that is required for them to perform their particular job function. As previously mentioned, a policy such as this can be implemented first by determining the information subset the user requires, then, (in the case of the QBE database) by identifying those base tables as views that contain the pertinent data and finally by granting access rights in accordance with these determinations.

Figure 1 illustrates a subset of the relational database [6] that would be used to define the

CUSTOMER	CUSTOMER #	LAST NAME	FIRST NAME	CREDIT	ADDRESS
	G._CN88	_JONES		_CL1	

CUSTOMER SALE	CUSTOMER #	TIME	DATE
	_CN88	_T1	_D1

SALE	TIME	DATE	AMOUNT	INVOICE
	_T1	_D1	ALL._A1	

SALE PAYMENT	SALE TIME	SALE DATE	PAYMENT TIME	PAYMENT DATE
¬	_T1	_D1		

ACCOUNTS RECEIVABLE	CUSTOMER NAME	AMOUNT OWED	CREDIT
I.	_JONES	SUM. ALL._A1	_CL1

Fig. 2. Program for Subsidiary Accounts Receivable.

appropriate views (in an internal control sense) for the customer transactions. On the left is the portion of the enterprise's conceptual schema that contains the data elements necessary to produce the separate functional views on the right. The conceptual model was formulated using an Entity-Relationship [1] overview in which each base relation or QBE table corresponds directly to either an entity set (box) or a relationship set (diamond).

Figure 2 illustrates the QBE derivation of the logical view of subsidiary accounts receivable. The top four relations (or tables) are base conceptual elements, (which correspond directly to the boxes and diamonds of Figure 1) while the bottom relation is a logical view used to output the data elements required by the accounts receivable clerk. The "CUSTOMER NAME" and "CREDIT" col-

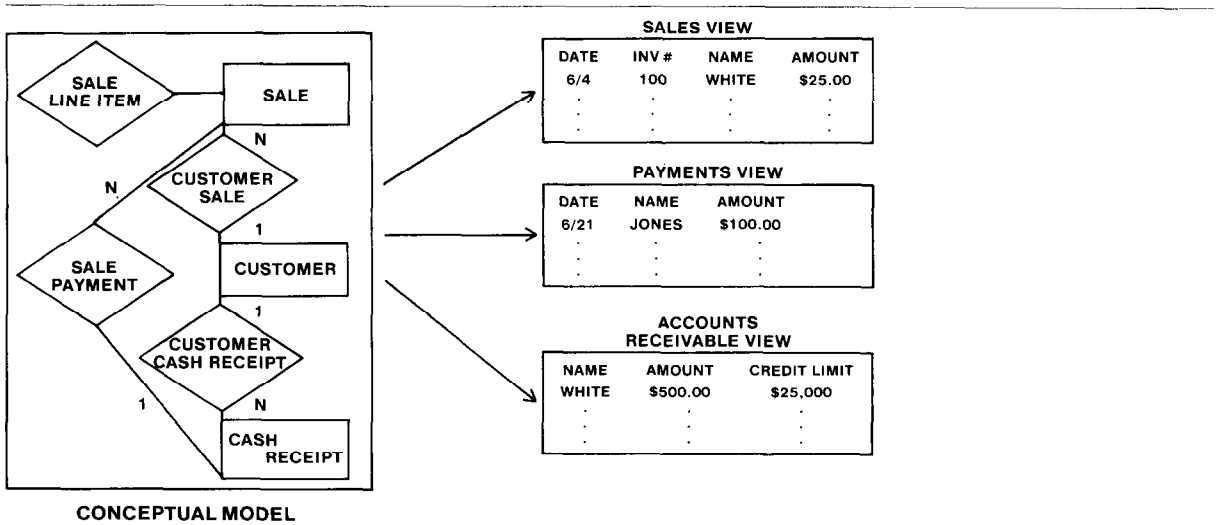


Fig. 1. Relational Database Subset.

umns are transferred directly from the "CUSTOMER" relation by placing the example elements, JONES and _CL1, in the appropriate columns of the source ("CUSTOMER") and destination ("ACCOUNTS RECEIVABLE") tables. The extension of the "AMOUNT OWED" column is derived by obtaining a total amount for unpaid customers sales (ALL._A1).

The total amount of sales for a particular customer is found by linking through the relationship table "CUSTOMER SALE" and obtaining the "TIME" (_T1) and "DATE" (_D1) entries for all sales to a particular customer (as identified by the "CUSTOMER NUMBER" example element _CN88). The query then uses the same "TIME" and "DATE" data elements to find the "SALE" table rows (or entries) for the customer, with a further condition on those sales. The portion of the query in the "SALE PAYMENT" table requires that entries in the "SALE" table to be used for the customer do not also appear as paid sales, i.e. the query says "give me all the sales for the customer for which payment has not been received". The result in _A1 will be the amount of unpaid sales or accounts receivable. [1]

The results of this query are inserted (as specified by the I. operator) as rows (or entries) in the "ACCOUNTS RECEIVABLE" table by customer (as stipulated by the G. or 'group by' operator). After the query has been formulated, it is now possible to construct the proper authority constraint over this particular logical view.

For proper specification of internal controls, particular subsets of the corporate data pool should only be available to the employees who require this view of the data to perform their job function. Because the QBE system is closed with respect to access rules, this means it is necessary to unlock this view to the person who requires the data. In a typical business organization, the employee who authorizes further credit sales needs that subset of the database which identifies those customers who have reached their credit limit. This person is also generally responsible for identifying those amounts which have become uncollectible and should be written off as such. For internal control purposes, this same employee should not have access to either the sales or payments data in disaggregate form, because such access would allow the employee to obtain the payments of accounts which were written off incorrectly. Figure 3 shows the procedure that would unlock this view to the employee whose job function is "CREDIT CLERK".

This authorization is formulated as a two-step query and is demonstrated in Figure 3(a). First, the employee table is accessed to produce the name of the employee whose job function is "CREDIT CLERK"; the result was the name "KATHY." The second portion of the query says to insert (I.) into the "AUTHORITY" [2] table to the user "KATHY" the ability to print (P.) the column entries in the table "ACCOUNTS RECEIVABLE." This authorization would be called a static authority constraint because it is valid only as long as "KATHY" is in the position of "CREDIT CLERK." When her job function changes, it would be necessary to reformulate the authority constraint with the name of the new employee in charge of credit. Figure 3(b) shows a dynamic representation of this authorization which is not available under the current commercial version of QBE but which is more desirable because it doesn't change as employees switch job functions.

Another job function that would require access to data elements from the conceptual schema represented in Figure 1 concerns the entry of sales into the database. In business organizations, this job is generally performed by the order entry clerk, and it entails adding or recording the occurence of sale events. For internal control purposes, this person should only have access to data element or tables that must be updated or augmented when a sale occurs. This person should not have access to payment information, as it would allow the clerk to neglect to record both the sale and payment transactions: a situation which allows the misappropriation of funds. Once again, to allow access to the logical view of the database required for sales entry, it is necessary to unlock

[1] This is essentially an implementation of a set difference operation. It works because the relationship between the sales and the payments is of the type $N:1$. This operation would not work if the relationship was of the type $M:N$. Later in the paper there is an example of a similar operation for this type of relation.

[2] The "AUTHORITY" table is a separate relation that is used by the system to keep track of constraints that have been established. Entries to this table are made by the system when queries are formulated with an I. AUTH as in Figures 3, 4 and 7.

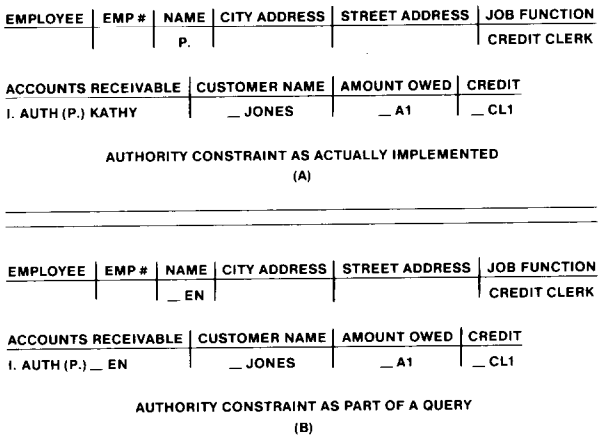


Fig. 3. Establishing Authority Constraints on the Accounts Receivable View.

the portion of the database that must be accessed to record sales transactions.

Examination of the conceptual schema in Figure 1 shows that "SALE" is represented as a table and that it is involved in three relationship tables: (1) "CUSTOMER SALE" which identifies the customer that the sale was made to, (2) "SALE PAYMENT" which relates the sale with the cash receipt that pays for the sale, and (3) "SALE LINE ITEM" which identifies the inventory for the sale. In order to process sales, it is necessary to add rows to the "SALE" table, the "CUSTOMER SALE" table, and the "SALE LINE ITEM" table, but not the "SALE PAYMENT" table. In fact, for internal control reasons, the "SALE PAYMENT" table should not be accessible to the person entering sales transactions.

Because each sale is made to only one party, each entry in the "SALE" table requires a single entry in the "CUSTOMER SALE" table. Each entry will consist of the time and date of the sale and the number of the customer to whom the sale was made. For each sale entry there will be "N" or many entries in the "SALE LINE ITEM" table. The entries in this table identify the inventory items that appear on the sale and will consist of the time and date of the sale, the inventory number, the quantity of the particular item sold, and the price actually charged.

When a sale transaction is entered, it would trigger [3] the system to require that appropriate entries be made to each of these tables to complete a sales transaction. Further, in order to maintain the integrity of the system, it would be necessary to check that the sum of the extensions for all "SALE LINE ITEM" entries equals the total amount of the sale. The QBE system initially described in [14] supports this type of integrity constraint.

Figure 4 shows the formulation of the authority constraints that would allow the "ORDER ENTRY" clerk access to the appropriate data elements. The authority constraint as actually implemented consists of two separate components and is shown in part (a) of the figure. The first part consists of a query on the "EMPLOYEE" table to find the person whose job function is "ORDER ENTRY." The result – employee "ANN" – is used as part of the query in the second part of the authority constraint. The second part inserts into the "AUTHORITY" table (for employee "ANN") the ability to insert (I.) entries in the rows of the "SALE", "CUSTOMER SALE" and "SALE LINE ITEM" tables. Figure 4(b) demonstrates a situation to be explained in more detail later: the formulation of this authority constraint in a dynamic environment.

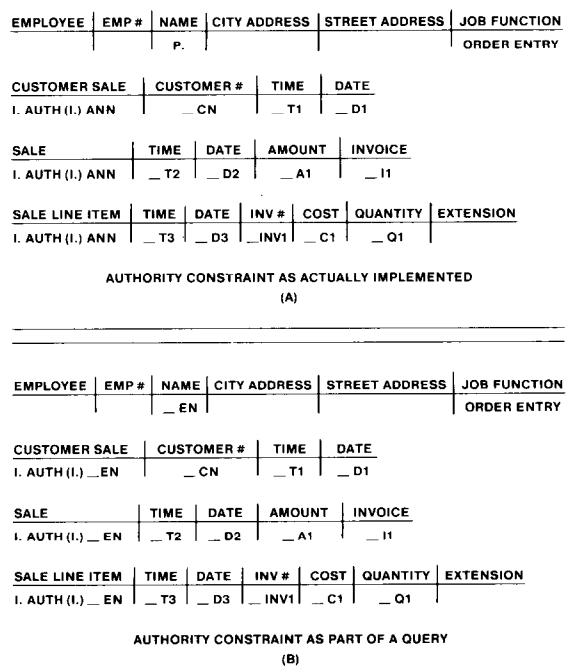


Fig. 4. Establishing Authority Constraints on the insertion of Sales Data.

[3] Triggers are essentially stored programs that are initiated when certain actions occur in the database. For a more complete discussion of this concept, see [10].

2.2. Accounts Payable for Services

This section demonstrates the formulation of the view of the database that would be consistent with the processing of cash disbursements to vendors that supply services to the company. This is very similar to the Accounts Receivable example except that the relationships between the initial event or transaction, ("Sale" and "General and Administrative Expense") and the payment transactions, ("Cash Receipt" for "Sale" and "Cash Disbursement" for "General Administrative Expense") are of different types (see Figures 1 and 5).

The relationship between Sale and Cash Receipt is many-to-one which means that one cash receipt from a customer will completely pay for a group of sales. This allows accounts receivable to be calculated by looking at the sales that were not part of the "SALE PAYMENT" table. In contrast, the General Administrative Expense-Cash Disbursement relationship has been identified as many-to-many. This means that one particular disbursement will pay for many general and administrative expense transactions and one particular transaction can be paid for by many disbursements, i.e. partial payments are allowed. Therefore, to calculate accounts payable, it is not sufficient to simply identify those general and administrative expense items that have not had a payment related to them, because it might be a partial payment. It is now necessary to add amounts of services received from vendors and subtract the amounts paid to get a correct total for accounts payable. Figure 5 shows the portion of the conceptual model that is needed to obtain the information for this particular subset of the total accounts payable. The formulation of the view "ACCOUNTS PAYABLE SERVICE" uses tables derived from this portion of the conceptual model to accomplish the task.

Figure 6 shows the structure of the query that provides the authorized employee the logical view of the database that corresponds to accounts payable for general and administrative services for each vendor. The query specifies that the result is to be presented in the "ACCOUNTS PAYABLE SERVICE" by "VENDOR #" as stipulated by the 'group-by' operator (G.) on the example element _VN. In addition the name of the vendor, identified by the example element _SMITH, is also to be placed in the "ACCOUNTS PAYABLE SERVICE" table. There are also two calculations that are being done as part of the query; first the total amount of general and administrative expenses for each vendor is obtained, and second the total amount of payments made to the vendor for these services is derived.

To calculate the total value of the services provided by a vendor the query uses the relationship table "GEN ADMIN SUPPLY" to link vendors (identified by _VN) with the time (_T1) and date

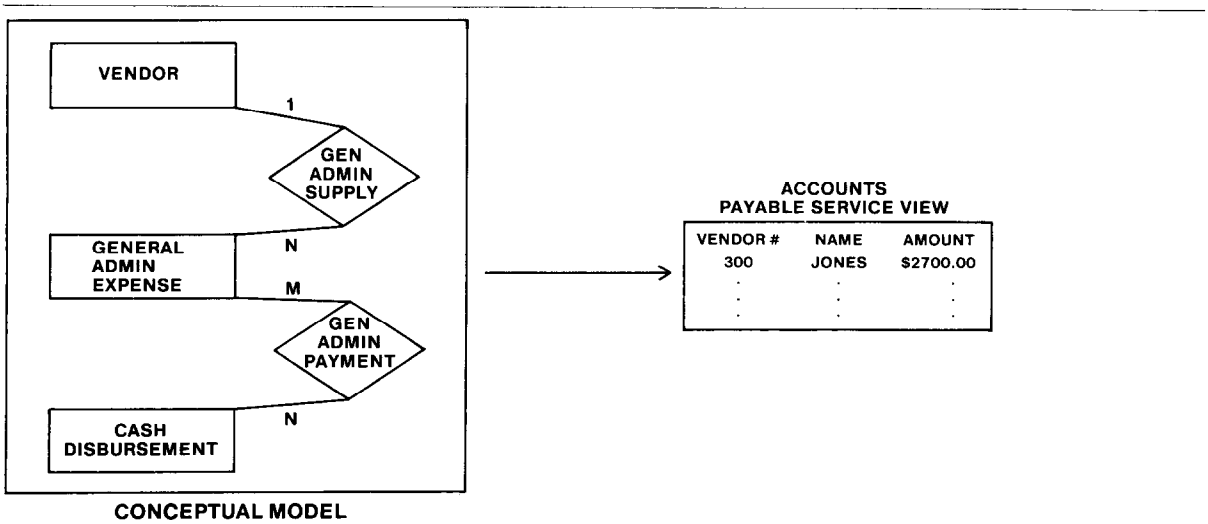


Fig. 5. Portion of Conceptual Model.

VENDOR	VENDOR #	VENDOR NAME	CITY ADDRESS	STREET ADDRESS
	G._VN	_SMITH		

GEN ADMIN SUPPLY	VENDOR #	SERVICE TIME	SERVICE DATE
	_VN	_T1	_D1
	_VN	_T2	_D2

GEN ADMIN EXPENSE	SERVICE TIME	SERVICE DATE	TYPE	AMOUNT
	_T1	_D1		ALL._A1

GEN ADMIN PAYMENT	SERVICE TIME	SERVICE DATE	PAYMENT TIME	PAYMENT DATE
	_T2	_D2	_T3	_D3

CASH DISBURSEMENT	PAYMENT TIME	PAYMENT DATE	CHECK #	VOUCHER	AMOUNT
	_T3	_D3			ALL._A2

ACCOUNTS PAYABLE SERVICE	VENDOR #	VENDOR NAME	AMOUNT
I.	_VN	_SMITH	(SUM. ALL._A1 - SUM. ALL._A2)

PROGRAM ACCOUNTS PAYABLE SERVICES

Fig. 6. Program Accounts Payable Services.

(_D1) of the service transactions they provided. Then these time and date entries are used to select the vendor entries in the "GEN ADMIN EXPENSE" table. For each of the rows (or entries) that were selected, the figure in the "AMOUNT" field is accumulated in ALL._A1. This means that when this portion of the query is executed the total amount of services provided by a vendor is stored in the example element ALL._A1.

The second calculation obtains the total amount paid to the same vendors for the services they provided (it will not include amounts paid to the vendor if they also provided inventory for example). Once again the "GEN ADMIN SUPPLY" table is use to identify the time (now_T2) and date (_D2) of the services provided by the particular vendor (_VN). The obtained time and date entries are now used in the relationship table "GEN ADMIN PAYMENT" to identify the time (_T3) and date (_D3) entries of the payments that were made for the particular "GEN ADMIN EXPENSE" transactions. The time and date entries are then used to obtain the rows of the "CASH DISBURSEMENT" table corresponding to these payments. Finally the figure in the "AMOUNT" field is accumulated in ALL._A2. The results of this portion of the query is the total dollar amount paid to a particular vendor for general and administrative services that were provided.

It is now possible to combine the results of these calculations in the table "ACCOUNTS PAYABLE SERVICE". The vendor's name (_SMITH) and number (_VN) fields are inserted directly from the "VENDOR" table. The entry in the "AMOUNT OWED" is the sum of the previous calculations (SUM.ALL._A1-SUM.ALL._A2 or services minus payments). The result of this query will be a list, by vendor, of those vendors

EMPLOYEE	EMP #	NAME	CITY ADDRESS	STREET ADDRESS	JOB FUNCTION
	P.				AP SERVICE

ACCOUNTS PAYABLE SERVICE	VENDOR #	VENDOR NAME	AMOUNT
I. AUTH (P.) BILL	_VN	_SMITH	_A1

AUTHORITY CONSTRAINT AS ACTUALLY IMPLEMENTED
(A)

EMPLOYEE	EMP #	NAME	CITY ADDRESS	STREET ADDRESS	JOB FUNCTION
		_EN			AP SERVICE

ACCOUNTS PAYABLE SERVICE	VENDOR #	VENDOR NAME	AMOUNT
I. AUTH (P.)_EN	_VN	_SMITH	_A1

AUTHORITY CONSTRAINT AS PART OF A QUERY
(B)

Fig. 7. Establishing Authority Constrains on Accounts Payable View.

who supply general and administrative services and the amounts that are owed to them.

The authority constraint for this logical view is demonstrated in Figure 7(a). The first part requires accessing the "EMPLOYEE" table to identify the name of the person whose job function is "AP SERVICE." The result is employee "BILL" and the second part of the authority query inserts (I.) into the "AUTHORITY" table the ability for "BILL" to print (P.) the entries in the table.

2.3. Section Summary

This section has demonstrated a method of implementing restriction in data access that would be consistent with separation of duties as required by necessary internal controls. This method was demonstrated in a database environment in which data was stored in a corporate-wide data pool and managed with the relational database management system QBE. The method requires specification of views consistent with particular job functions and then assertion of authority constraints.

In the next section some of the limitations of this implementation and the QBE DBMS are discussed.

3. Limitations of the Implementation

3.1. Dynamic Authority Constraints

In the three examples used in the paper the formulation of the authority constraints was static, i.e., valid only for a certain period of time. In an actual implementation, this would mean that every time an employee changes jobs, it would be necessary to identify the authority that person had, eliminate those occurrences not consistent with the new function and then insert new authority constraints. In the real-time environment of a typical business, this could cause inefficient use of the system. In Figures 3(b), 4(b) and 7(b), dynamic formulations of the authority constraints are displayed.

In the static forms, it would only be necessary to access the "AUTHORITY" table to see if the attempted query would be allowed; this would not be the case with the dynamic constraints. The dynamic constraints are not established for a particular person (such as ANN, BILL or KATHY). Instead, they are established for particular job functions ("CREDIT CLERK", "ORDER ENTRY" or "AP SERVICE"). Thus after the appropriate views had been formulated for certain functions, authority to use these views would not need to change as employees change jobs. This would mean that, before a certain query is allowed, the "AUTHORITY" table would have to be accessed and then the "EMPLOYEE" table would have to be checked to identify the employee performing the particular job function at that time.

The dynamic authority constraints represented in the figures use the example element _EN in place of a specific name. When the authority constraint is checked, the QBE database management system must access the "EMPLOYEE" table to find the appropriate name. The use of this example element rather than a specific name makes these formulations *dynamic* authority constraints. These types of dynamic constraints were described in a preliminary paper concerning security and integrity aspects of the QBE system [14]; however, they were not part of the commercial package. It is our opinion that these dynamic constraints would be a very desirable feature of any future relational database management system that would be used in an accounting environment.

3.2. Integrity Constraints

Integrity constraints are conditions of the database that can be stated apriori to be necessarily true. These constraints ensure that certain properties of the data will be checked and/or maintained on an ongoing basis. A good example of this type of constraint can be demonstrated using the logical view for sales processing.

As previously mentioned, the employee in charge of entering sales transactions would invoke a program or procedure that would provide the tables necessary to enter sales. This would include; (1) add a row to the "SALE" table, (2) make an entry to the "CUSTOMER SALE" table consisting of the customer number and the time and date of the sale [4] which will link the customer to the

[4] This is a method of constructing a relationship between two entities, i.e. building a separate relation that has the key fields of the entities as its attributes. In this case, the customer number is the key of customer and the time and date attributes serve as the key of the sale transaction.

sale, and (3) add a row to the "SALE LINE ITEM" table to link the sale with the inventory items of the sale, (which means an entry for each inventory item). Each entry to the "SALE LINE ITEM" table will consist of the time and date of the sale, the inventory number, the price charged and the quantity of the item. In addition the extension (price times quantity) is included which will be used to maintain the condition that the sum of these extensions equal the amount of the sale.

Figure 8(a) demonstrates an implementation of the integrity constraint that would ensure the accuracy of any sale that is added to the database. The example elements _T1 ("TIME") and _D1 ("DATE") are used to link the sale with the line-item occurrences for that particular sale and the extensions are stored in _A1 (.ALL._A1). The constraint is inserted (I.) in the "SALE" table and states that the entry in the "AMOUNT" field must be equal to the sum of the extensions (SUM.ALL._A1). The constraint is to be checked only upon insertion [CONSTR(I.)] of entries to the "SALE" table. Due to the fact that entries to the "SALE" table require entries to the "SALE LINE ITEM" table, this amounts to a check anytime an entry is made to either table.

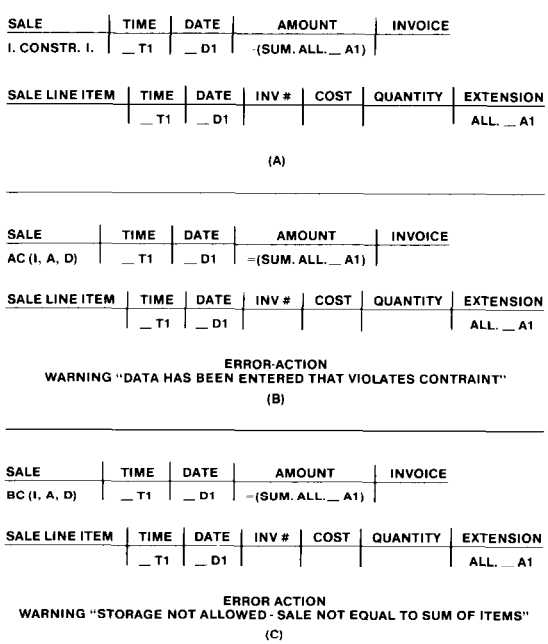


Fig. 8. Semantic Integrity Constraints.

There are other approaches to the specification of integrity constraints besides that described by Zloof [14]. In the paper by Theerachetmongkol and Montgomery [11], integrity constraints are specified not only according to the operation which would initiate the checking of the constraint (I insert, D delete, A amend or U update) but also the sequence of the check. The authors describe three types of constraint checking sequences, perpetual (PC), pre-operative (BC or BA) and post-operative (AC, AA or AR). The integrity constraint in this example would belong to either of the operative class of constraints, i.e., it is only necessary to check the constraint in connection with a storage operation.

If the constraint were specified as in Figure 8(b), the type of constraint would be a post-operative check (AC). This would tell the system to perform the storage operation (add a sale transaction to the database) and then check that the sum of the extensions equals the amount of the sale. If the constraint is violated, then the error return would be taken and the message would be displayed requesting that corrective action be taken. However, until the corrective action was taken, the integrity of the system would be violated, i.e., the sum of the parts of sales would not equal the sum of all sales. Therefore a query that used this portion of the data would not necessarily yield correct responses. This problem could be alleviated by formulating the constraint in the pre-operative format (BC).

Figure 8(c) shows a pre-operative (BC) formulation of the previously mentioned constraint. In this case the constraint is expressed identically to the post-operative constraint except that the constraint would be checked prior to the completion of any storage operation. By formulating constraints in this manner the database will always contain data that has the predetermined integrity requirements. The problem with this solution is that any query on this subset of the data will not necessarily have access to the most current data elements. Depending on the cause of the integrity violation, the correction could require an investigation that could keep the database from being current for an unacceptable length of time.

The selection of the type of integrity constraint [pre- or post-operative] will of course depend on the nature of the uses for which this particular subset of the database is required.

4. Conclusion

As the facilities that are used to implement a particular information system become more complex, the ability for managers to personally exert control over the data declines dramatically. This inability to control the data is particularly a problem in the shared data environment of a DBMS. If these DBMS are to be used in a business environment the system must become part of the process that controls the access to, and integrity of, the data.

This paper has demonstrated an implementation of authority constraints using the data dictionary facilities of a relational DBMS, Query-by-Example. The constraints were formulated to correspond to information subsets that are required by particular job functions within a typical business organization. The paper went on to discuss the formulation of integrity constraints on this same subset of data.

The paper suggests two avenues for future work. The first direction concerns implementation of these types of constraints using other dictionary facilities that are available. The other direction would be to look at internal controls not at the implementation level, but at the level of the overall data model of the firm. This would allow all of the constraints on the data to be expressed in the overall semantic representation of the enterprise.

References

[1] P.P. Chen, "The Entity-Relationship Model – Toward a Unified View of Data." *ACM Transactions on Database Systems*. Vol. 1 (March 1976), pp. 9–36.

[2] B.E. Cushing. *Accounting Information Systems and Business Organizations* Reading, MA.: Addison-Wesley Publishing Co., 1982.

[3] C.J. Date. *An Introduction to Database Systems* (3rd ed.) Reading, MA.: Addison-Wesley Publishing Co., 1977.

[4] E.B. Fernandez, R.C. Summers and C. Wood. *Database Security and Integrity*. Reading, MA.: Addison-Wesley Publishing Co., 1981.

[5] G. Gal and W.E. McCarthy. "Declarative and Procedural Features of a CODASYL Accounting System." in *Entity-Relationship Approach to Information Modeling and Analysis*. P.P. Chen ed. Amsterdam.: North-Holland Publishing Co., 1983.

[6] G. Gal and W.E. McCarthy, "Operation of a Relational Accounting System." Working Paper, Michigan State University, September 1983.

[7] IBM, *Query-by-Example Program Description/Operation Manual*. SH20-2077-2. (October 1980), White Plains, N.Y.

[8] W.E. McCarthy. "An Entity-Relationship View of Accounting Models." *The Accounting Review*, Vol. 54 (October 1979), pp. 667–686.

[9] W.E. McCarthy. "Multidimensional and Disaggregate Accounting Systems: A Review of the 'Events' Accounting Literature." *MAS Communication*. Vol. 5 (July 1981), pp. 7 13.

[10] W.E. McCarthy. "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment." *The Accounting Review*. Vol. 57 (July 1982) pp. 554–578.

[11] A. Theerachetmongkol and A.Y. Montgomery. "Semantic Integrity Constraints in the Query-by-Example Data Base Management Language." *The Australian Computer Journal*, Vol. 12 (February 1980) pp. 28–42.

[12] J.D. Ullman. *Principles of Database Management*. Potomac, Maryland: Computer Science Press, Inc., 1983.

[13] M.M. Zloof. "Query by Example." *Proceedings of the NCC* (May 1975).

[14] W.E. McCarthy "Security and Integrity Within the Query-by-Example Data Base Management Language." IBM Research Working Paper, RC6982, February 1978.