

Logical Design of Audit Information in Relational Databases

Sushil Jajodia, Shashi K. Gadia, and Gautam Bhargava

In the “Trusted Computer System Evaluation Criteria” [DOD85], the accountability control objective is stated as follows:

Systems that are used to process or handle classified or sensitive information must assure individual accountability whenever either a mandatory or discretionary security policy is invoked. Furthermore, to assure accountability the capability must exist for an authorized and competent agent to access and evaluate accountability information by a secure means, within a reasonable amount of time and without undue difficulty.

Existing databases clearly fail to meet this objective. They treat only the current data in a systematic manner; the old information is either deleted or stored on an ad hoc basis. If we wish to meet the above requirements in a database, we need to treat the audit data in a systematic way as well, not just the current data. Once this is done, we can then begin to address other issues related to audit.

A similar situation existed in the early 1960s. An application program used its own specially designed files. As a consequence, it was very difficult for a user to know what files already existed in the system. Knowing about the existence of a file was not sufficient; the user needed to know the actual file structure as well. If a file maintained by some program was reorganized, there was no assurance that other programs wishing to access the reorganized file would still work. Thus, much information was stored redundantly; however, there were problems when users started to look for mutual consistency of replicated data.

The database approach in the early 1970s overcame many of these problems. An important tool called a *data model* was developed, which

imposed a logical structure on all the (current) data in the system. This allowed users to see data not as an arbitrary collection of files, but in more understandable terms. Database researchers developed another key concept called *independence*: The logical structure of the data became independent of the details of physical storage of data. Since now users could reorganize the physical scheme without changing the logic of existing programs, duplication of data could be avoided.

We assert that we must take a similar approach when it comes to audit data. First we must carefully list the audit objectives, then provide mechanisms to achieve these objectives. In addition to the actual recording of all events that take place in the database, a logical structure needs to be imposed on audit data. An audit trail requires mechanisms for a complete reconstruction of every action taken against the database [BJOR75, BONY88, FERN81]. Finally, an audit trail must also provide the capability (a query language) to easily access and tools to evaluate the audit information.

The organization of this essay is as follows. First we give the audit requirements and argue the need for two time dimensions to logically organize audit data. Then we present some notation and definitions. We devote a section to the database activity model and another to showing how we can restrict access to the database.

Audit requirements

A detailed discussion of audit requirements in trusted systems has been published by the National Computer Security Center [NCSC88] and other discussions are available in the literature [BJOR75, BONY88, FERN81]. An audit trail is a mechanism for a complete reconstruction of every action taken against the database: *who* has been accessing *what* data, *when*, and in what *order*. Thus, it has three basic objects of interest:

- The user. Who initiated a transaction, from what terminal, and when?
- The transaction. What was the exact transaction that was initiated?
- The data. What was the result of the transaction? What were the database states before and after the transaction initiation?

In addition to the actual recording of all events that take place in the database, an audit trail must also provide query support for auditing. The central point of this essay is to describe the data model called the *database activity model* [JAJO90g], which meets these objectives by providing a convenient mechanism for recording and querying accesses to the database.

Need for two time dimensions

In an auditable system, there must be a way to examine an old value of an attribute and to allow correction to this old value. At the same time, the database should retain the old value for future query and re-examination. This requires at least two time dimensions: one time dimension to order every operation against an object and a second time dimension to time stamp values of objects with their periods of validity in the real world [BJOR75, JAJ090g, LUM87, SNOD86, SNOD87]. This second time dimension is different from the first time dimension since an earlier value of an object may be corrected later in time.

As an example, suppose we have a relation called EMPLOYEE having two attributes NAME and SALARY. Suppose we inserted the tuple (Smith, 25K) on 1/11/85. On 3/2/86, we discovered an error in Smith's salary and made a retroactive change to 30K. Now, in an auditable system such as we discuss here, it is possible to overwrite errors, but records will be kept of any errors that are corrected. Thus, we need to retain both facts in the database:

1. Smith's salary was known to be 25K from 1/11/85 to 3/2/86.
2. On 3/2/86, a change was made to the salary from 25K to 30K, and this change was retroactive.

If we simply change the salary from 25K to 30K, we lose the second fact that an error was discovered in Smith's salary on 3/2/86, a change was made, and this change was retroactive. On the other hand, if we just change the value of the first time parameter from 1/11/85 to 3/2/86, we lose the first fact that the salary was considered to be 25K from 1/11/85 to 3/2/86.

Therefore, we need two different time parameters to describe this situation. We call the first time dimension the *transaction time*, the time at which the information is stored in the database. The second time dimension is the *valid time*, the time when the information in the database is valid in the real world. A relation that incorporates both transaction and valid times is called a *bitemporal relation*.

Preliminaries

We work in the context of the relational model of data. We presume the reader is familiar with relational database theory. A *relation scheme* R is a collection of attributes. K is said to be a *key* of R if the functional dependency $K \rightarrow R$ holds and if K does not contain a proper subset K' such that $K' \rightarrow R$ holds. A *relational scheme* \mathbf{R} is a collection $\{R_1, \dots, R_n\}$ of relation schemes, and a database \mathbf{r} over \mathbf{R} is a set of relations $\{r_1, \dots, r_n\}$ such that each r_i is a relation over R_i .

We assume that users access the database by executing procedures, called *transactions*. A transaction is a sequence of operations viewed as an atomic unit of integrity, consistency, and recovery. We allow the usual operations in any transaction:

- Insert. Add a tuple in a relation.
- Delete. Remove a collection of tuples from a relation.
- Modify. Change values of a tuple in a relation.
- Retrieve. Access all tuples satisfying a given condition.

The database activity model

In this section, we describe the database activity model [JAJ090g]. A formal treatment of the temporal aspects of our model is given elsewhere [BHAR90, BHAR93].

Let \mathbf{R} be a relational scheme. The *database activity scheme* over \mathbf{R} is a triple $\langle \mathbf{D}, \mathbf{Update-Store}, \mathbf{Query-Store} \rangle$ where

- For every relation scheme R in \mathbf{R} , there is a two-dimensional temporal relation D (as illustrated in Figure 1) in \mathbf{D} . It encapsulates the complete history of each and every modification made to a value of an attribute in R .
- For every D in \mathbf{D} , there is a relation $Update-Store(D)$ in $\mathbf{Update-Store}$. $Update-Store(D)$ records the circumstances surrounding the updates made to D .
- $Query-Store$ is a single relation. It is essentially a log of all queries.

We describe each of these in detail next.

The bitemporal relations. We will describe a temporal relation that incorporates the concept of time at the logical level of design and use it to represent the two time dimensions. The resulting relation is called a *two-dimensional temporal relation*. An example of a bitemporal relation is shown in Figure 1. Although time typically consists of times or dates of events (as shown in Figure 3 later), for simplicity we will use integers as time values.

In a bitemporal relation, two different time parameters are associated with each value:

- *Transaction time*. The first time value, called the transaction time, is used to capture the complete history of all operations on an object. The domain for the transaction time is the time interval $[0, now)$, where *now* is a special symbol that denotes the changing value of the present time.

- *Valid time.* The second time value, called the valid time, is used to time stamp a value with its period of validity in the real world, giving rise to a historical relation. The domain for the valid time is the time interval $[0, \infty)$. We augment $[0, \infty)$ with a new symbol uc , which stands for “until changed” [WIED91]. When we associate the valid time $[l, uc)$ with a value, the value is interpreted to be valid until it is changed.

NAME	SALARY	DEPT
$[8, now) \times [11, uc)$ John	$[8, 53) \times [11, uc)$ 15K $[53, now) \times [11, 50)$ 15K $[53, now) \times [50, uc)$ 20K	$[8, 40) \times [11, uc)$ Toys $[40, now) \times [11, 45)$ Toys $[40, now) \times [45, uc)$ Shoes
$[48, now) \times [48, uc)$ Doug	$[48, now) \times [48, \infty)$ 20K	$[48, now) \times [48, uc)$ Auto

Figure 1. A bitemporal relation.

Thus, our changing knowledge of the real world is modeled by associating two-dimensional time stamps with each value, and $[0, now) \times [0, \infty)$ serves as the universe of time. It should be pointed out that while the transaction time is always monotonically increasing, the valid time is not, since our knowledge of a history may change with time, requiring updates to the historical database.

Figure 2 illustrates how the two-dimensional time stamps should be interpreted. It shows the values taken by the attribute SALARY for the employee John in Figure 1.

$[8, 53) \times [11, uc)$ 15K
$[53, now) \times [11, 50)$ 15K
$[53, now) \times [50, uc)$ 20K

Figure 2. John’s salary in the bitemporal relation.

The semantics of the values is that “there was a transaction posted at time 8 declaring John’s salary to be 15K from time 11 onward, but another transaction that was posted at time 53 updated John’s salary to be 15K from time 11 to 50 and 20K from time 50 onward.” Thus, we can think of salary as consisting of many versions, and the transaction time

reflects the sequence of changes made to the salary over time. The valid time associated with each version provides the period of validity of that version.

Figure 3 gives the bitemporal relation for our earlier example with Smith. It is easy to verify that the temporal relation given in Figure 3 cleanly and accurately models the situation we described. It shows that Smith's salary is 30K from 1/11/85 onward; however, Smith's salary was known to be 25K during the time period 1/11/85 to 3/2/86, and a correction was made to the salary on 3/2/86.

NAME	SALARY
$[1/11/85, \textit{now}) \times [1/11/85, \textit{uc})$ Smith	$[1/11/85, 3/2/86) \times [1/11/85, \textit{uc})$ 25K $[3/2/86, \textit{now}) \times [1/11/85, \textit{uc})$ 30K

Figure 3. The bitemporal relation for the Smith example.

Thus, we see that the reexamination of historical data is possible in a bitemporal relation, and this allows corrections to be made as necessary (but still retains for future query and examination the fact of and data associated with the original “estimate” or “observation”). Moreover, the database may carry predictions of future operations or values. For example, the data may contain the required list of actions for the following weeks, such as times and places of any meetings, personnel actions to be taken, and so on [WIED91].

Update-Store and Query-Store relations. For each bitemporal relation, the database activity model maintains a relation called an *Update-Store* relation, which records information about all updates to the relation, and a relation called a *Query-Store* relation, which is essentially a log of all queries concerning the relation.

More formally, let R be a relation scheme in \mathbf{R} , and let K be the key for R . Then the attributes of *Update-Store*(R) are the key K , the transaction time (TT), the authorizer of the transaction (AUTHORIZER), the user of the transaction (USER), and the reason for executing the transaction (REASON). (See Figure 4 below.)

The *Query-Store* relation contains a log of all the queries. It has the following attributes: the query (QUERY), the query time (TT), and the user posing the query (USER). (See Figure 5.)

One interesting property is that once we supplement the bitemporal relations with the *Update-Store* and *Query-Store* relations, the transaction log can be restored from these three relations. There is never any

loss of information, and therefore we have the complete audit trail concept. We illustrate this next by way of an example.

Suppose our database consists of a single relation scheme EMP with attributes NAME, SALARY, and DEPT such that the attribute NAME is the key. Suppose for each transaction, we wish to keep track of the following audit information: transaction time (TT), who authorized the update (AUTHORIZER), the user making the update (USER), and the reason for the update (REASON). For each query, we store the query (Query-id), time of the query (TT), and user making the query (USER). Consider the following sequence of transactions:

- T₁. TT = 8; User = Mark.
insert (NAME: [11, *uc*] JOHN; SALARY: [11, *uc*] 15K; DEPT: [11, *uc*] Toys)
with (Authorizer = Don; Reason = New Employee);
- T₂. TT = 40; User = Ryne.
modify (NAME: [11, *uc*] JOHN) to (DEPT: [11, 45] Toys, [45, *uc*] shoes)
with (Authorizer = Don; Reason = Reassignment);
- T₃. TT = 42; User = Vance.
Q1: What is John's salary?
- T₄. TT = 48; User = Rick.
insert (NAME: [48, *uc*] Doug; SALARY: [48, *uc*] 20K; DEPT: [48, *uc*] Auto)
with (Authorizer = Joe; Reason = New Employee);
- T₅. TT = 53; User = Dameon.
modify (NAME: [11, *uc*] JOHN) to (SALARY: [11, 50] 15K, [50, *uc*] 20K)
with (Authorizer = Don; Reason = Promotion);
- T₆. TT = 54; User = Andre.
Q1: What is John's salary?
- T₇. TT = 55; User = Mitch.
Q2: What is John's department?
- T₈. TT = 56; User = Don.
Q3: Who made inquiries about John's salary?
- T₉. TT = 58; User = Paul.
Q2: What is John's department?

Figure 1 gives the two-dimensional temporal relation corresponding to these transactions. Figures 4 and 5 give the resulting Update-Store and Query-Store relations, respectively.

NAME	TT	AUTHORIZER	USER	REASON
John	8	Don	Mark	New Employee
John	40	Don	Ryne	Reassignment
Doug	48	Joe	Rick	New Employee
John	53	Don	Dameon	Promotion

Figure 4. The Update-Store relation.

QUERY	TT	USER
Q1: John's SALARY	42	Vance
Q1: John's SALARY	54	Andre
Q2: John's DEPT	55	Mitch
Q3: USER ID of Q1	56	Don
Q2: John's DEPT	58	Paul

Figure 5. The Query-Store relation.

It is easy to verify that all the transactions can be completely restored using the three relations in our database activity model: The values of the key NAME in the bitemporal relation (Figure 1) and in the Update-Store relation (Figure 4) set up a logical correspondence between the tuples in the two relations. By including the transaction time, the logical correspondence can be refined to a one-to-one correspondence between all the updates to the temporal relation and the tuples in the Update-Store relation. As a result, the transactions T_1 , T_2 , T_4 , and T_5 can be completely restored from the relations in Figures 1 and 4. The transactions T_3 and T_6 through T_9 can be completely restored from the relations in Figures 1 and 5. Finally, using the transaction time TT, we can order all nine transactions to obtain the original sequence of transactions.

Support for auditing. Another nice feature of the database activity model is that it has a simple yet powerful relational algebra to express audit and other queries about relations. We refer the reader elsewhere

for a brief description [JAJ090g] of the relational algebra and more complete descriptions [BHAR90, BHAR93].

Restricting access to the database

Since our database activity model contains information about the complete activity in the database system, we next consider how we can restrict user access to the database.

Assigning security level to transactions. It is possible to assign a security level to each transaction, which allows the incorporation of still further concepts of tagging the data with its security level at the object level [JAJ090g]. The data takes on the security level of the transaction because of the level of authority implied by the transaction creating or modifying the elements of the object. Thus, an object is itself not classified, but its parts carry different (possibly time-variant) security constraints. As a consequence, the response to a query or update transaction depends on the security level of the system user or the level at which the user is operating, in conjunction with the security policy that applies at the time of the operation.

The user hierarchy. A second way we limit access of a user is by filtering the database through a user hierarchy defined as follows. We use the symbol *now* to denote the changing value of the current time.

The master user. The master user has access to the whole database (for example, Figure 1) and enjoys the power to query errors and updates in the database.

The history user. This user sees only information filtered through the time domain $now \times [0, uc)$, and thus has access to only the most up-to-date knowledge of the history of objects (for example, the relation in Figure 6). Errors that have been corrected are not available to the history user. Such a user can ask questions of a historical nature such as “When did Tom’s salary increase?”

NAME	SALARY	DEPT
[11, <i>uc</i>) John	[11, 50) 15K [50, <i>uc</i>) 20K	[11, 45) Toys [45, <i>uc</i>) Shoes
[48, <i>uc</i>) Doug	[48, <i>uc</i>) 20K	[48, <i>uc</i>) Auto

Figure 6. The bitemporal relation as seen by a history user.

The snapshot user. The filter in this case is the time domain $now \times now$, and this user sees precisely what is available to a traditional user in databases (for example, Figure 7). In our framework such a user lies at the lowest level of the user hierarchy.

NAME	SALARY	DEPT
John	20K	Shoes
Doug	20K	Auto

Figure 7. The bitemporal relation as seen by a snapshot user.

The audit user. This user sees the data filtered through the time domain $\{(t, t') : t' \leq t\}$, does not have the concept of future (since $t \leq now$), and sees only the actions taken by the organization. Since the rest of the world is affected only by the organization's actions, this user can deal with the public relations and legal aspects of the enterprise.

The rollback snapshot user. This user sees the data filtered through the time domain $t \times t'$ for fixed values of t and t' . Like the snapshot user, this user sees a snapshot relation, with the difference that this user can see a snapshot at any point in the past, present, or future. For example, a rollback snapshot user will be given the snapshot in Figure 8 for the time domain 8×11 , and the snapshot in Figure 9 for the time domain 48×50 .

NAME	SALARY	DEPT
John	15K	Toys

Figure 8. The bitemporal relation filtered through the time domain 8×11 .

NAME	SALARY	DEPT
John	15K	Shoes
Doug	20K	Auto

Figure 9. The bitemporal relation filtered through the time domain 48×50 .

Conclusion

In an audit trail, it should be possible to audit every event in a database; we call this *zero-information loss*. What events are actually audited depends on the sensitivity of the events in question and the results of a careful risk analysis.

Our new model seems to be a building block for a general-purpose database system that holds the promise for a perfect logical organization of past, present, and future data. This aspect of our model can be exploited further to allow automatic regeneration and review of events that occur in the database system to detect and discourage security violations. As an example, Meadows and Jajodia [MEAD88b, Section 3] describe a situation with several instances where security requirements are in conflict with the data availability requirements, and they recommend auditing as a means of controlling information leaks. Our model can help identify such attempts to bypass security controls.