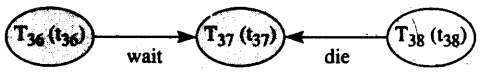
Figure 12.23 Example of wait-die deadlock prevention scheme.



Wait-Die

One solution in a case of contention for a data-item is as follows:

- If the requesting transaction is older than the transaction that holds the lock on the requested data-item, the requesting transaction is allowed to wait.
- If the requesting transaction is younger than the transaction that holds the lock on the requested data-item, the requesting transaction is aborted and rolled back.

This is called the **wait-die** scheme of deadlock prevention.

If concurrent transactions T_{36}, T_{37}, and T_{38} (having timestamp values of t_{36}, t_{37}, and t_{38}, respectively, with $t_{36} < t_{37} < t_{38}$) have at some instance a wait-for graph, as shown in Figure 12.23, then transaction T_{36} would be allowed to wait, but transaction T_{38} would be aborted and rolled back.

Wound-Wait

An opposite approach to the wait-die scheme is called the **wound-wait** scheme. Here the decision whether to wait or abort is as follows:

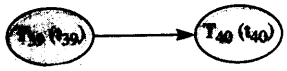
- If a younger transaction holds a data-item requested by an older one, the younger transaction is the one that would be aborted and rolled back (the younger transaction is wounded by the older transaction and dies!).
- If a younger transaction requests a data-item held by an older transaction, the younger transaction is allowed to wait.

For the request shown in Figure 12.24, where transaction T_{39} has a smaller timestamp value than transaction T_{40}, the younger transaction T_{40} would be aborted and rolled back, thus freeing the data-item locked by it to be used by transaction T_{39}.

For the request shown in Figure 12.25, where transaction T_{41} has a smaller timestamp value than transaction T_{42}, the younger transaction T_{42} is allowed to wait for the completion of the older transaction T_{41}.

We observe that in neither the wait-die scheme nor the wound-wait scheme is it required to abort and roll back an older transaction. In this way the older transaction

Figure 12.24 Example of wounding request.



cution schedule. If the precedence graph is acyclic, the schedule is serializable, which means that the database will have the same state at the end of the schedule as some serial execution of the transactions.

The concurrency control scheme ensures that the schedule that can be produced by a set of concurrent transactions will be serializable. One of two approaches is usually used to ensure serializability: delaying one or more contending transactions, or aborting and restarting one or more of the contending transactions. The locking protocol uses the former approach. Timestamp-based ordering, optimistic scheduling, and the multiversion technique of concurrency control use the latter.

In the locking protocol, before a transaction can access a data-item, it is required to lock the data-item in an appropriate mode. It releases the lock on the data-item once it no longer needs it. In the locking scheme, the two-phase locking protocol is usually used. The principle characteristic of the two-phase locking protocol is that all locks are acquired before a transaction starts releasing any locks. This ensures serializability; however, deadlock is possible.

With hierarchically structured storage of the database and its data-items, a different granularity of locking is implied. Thus, locking an item may imply locking all items that are its descendants. To enhance the performance of a system with hierarchically structured data, additional modes of locking are introduced. Thus, in addition to read and write locks, intention locks are required. The locking protocol is modified to require a root-to-leaf direction of lock requests and the reverse direction of lock releases.

In timestamp-based ordering, each transaction is assigned an unique identifier, which is usually based on the system clock. This identifier is called a timestamp and the value of the time-stamp is used to schedule contending transactions. The rule is to ensure that a transaction with a smaller timestamp (older) is effectively executed before a larger (younger) transaction. Any variation from this rule is corrected by aborting a transaction, rolling back any modifications made by it, and starting it again.

In optimistic scheduling, the philosophy is that a contention between transactions will be very unlikely and any data-item used by a transaction is not likely to be used for modification by any other transaction. This assumption is valid for transactions that only read the data-item. If this assumption is found to be invalid for a given transaction, the transaction is aborted and rolled back.

In the multiversion technique, data is never written over; rather, whenever the value of a data-item is modified, a new version of the data-item is created. The result is that the history of the evolution of a data-item is maintained. A transaction is assigned an unique timestamp and is directed to read the appropriate version of a data-item. The write operation of a transaction, such as T, could cause a new version of the data-item to be generated. However, in case another transaction has already produced a new version of the data-item based on the version used by transaction T, an attempt to write a modified value for the data-item by transaction T causes transaction T to be aborted, rolled back, and restarted as a new and younger transaction.

Deadlock is a situation that arises when data-items are locked in different order by different transactions. A deadlock situation exists when there is a circular chain of transactions, each transaction in the chain waiting for a data-item already locked by the next transaction in the chain. Deadlock situations can be either avoided or detected and recovered from. One method of avoiding deadlock is to ask for all data-items at one time. An alternative is to assign a rank to each data-item and request

locks for data-items in a given order. A third technique depends on selectively aborting some transactions and allowing others to wait. The selection is based on the timestamp of the contending transactions, and the decision as to which transactions to abort and which to allow to wait is determined according to the preemptive protocol being used. The wait-die and the wound-wait are two such preemptive protocols.

Deadlock detection depends on detecting the existence of a circular chain of transactions and then aborting or rolling back one transaction at a time until no further deadlocks are present. The wait-for graph is generated periodically by the system to enable it to detect a deadlock.

Key Terms

starvation
livelock
schedule
lost update
inconsistent read
phantom phenomenon
serial execution
serializable schedule
precedence graph
acyclic graph
cyclic graph
read-before-write protocol
concurrency control
lock
locking
lock manager
exclusive lock
shared lock
two-phase locking
growing phase
contracting phase
granularity
intention mode
intention share mode
intention exclusive mode
share and intention exclusive mode
tree-locking protocol
directed acyclic graph (DAG)
timestamp ordering
write timestamp
read timestamp
cascading rollback
optimistic scheduling
read phase
validation phase
write phase
multiversion
time-domain addressing
relative-most-recent version
wait-for graph
wait-die
wound-wait

Exercises

12.1 Consider two transactions as follows:

Transaction 1: $Fac\_Salary_i := 1.1 * Fac\_Salary_i + 1025.00$

Transaction 2: $Average\_Fac\_Salary := \sum_{i=1}^{N} Fac\_Salary_i/N$

What precaution, if any, would you suggest if these were to run concurrently? Write a pseudocode program for these transactions using an appropriate scheme to avoid undesirable results.

12.2 Consider that the adjustment of salary of the faculty members is done as follows, where Fac_Salary_i represents the salary of the ith faculty member:

Transaction 1: $Fac\_Salary_i := Fac\_Salary_i + 1025$

Transaction 2: $Fac\_Salary_i := Fac\_Salary_i * 1.1$

What precaution, if any, would you suggest if these were to run concurrently? Write a pseudocode program for these transactions using an appropriate scheme to avoid undesirable results

12.3 Consider the schedule of Figure 12.8a. What is the value of A and B, if $f_1(A)$ is $A + 10$, $f_2(B)$ is $B * 1.2$, $f_3(B)$ is $B = 20$, and $f_4(A)$ is $A * 1.2$? Assume that the initial values of A and B are 1000 and 200, respectively.

12.4 Repeat Exercise 12.3 for the schedule of Figure 12.9a.

12.5 Consider the transactions of Figure 12.21. Rewrite the transactions using the two-phase protocol and produce a schedule that is serializable.

12.6 Write an algorithm to find a cycle in a precedence graph. (Hint: Use an approach similar to that of algorithm 12.1)

12.7 Consider the transactions of Figure 12.21 and the schedule of Figure I. What would happen at step 5 if $t_{22} > t_{23}$? Complete the schedule after step 5 and give the values for A and B after each step. Assume that the initial values are A: {400, W_a, R_a} and B: {500, W_b, R_b}.

12.8 Given the following schedule of Figure M, in a system where timestamp ordering is used, suppose transactions T_{22} and T_{23} had been assigned timestamps t_{22} and t_{23} respectively and *Sum* is a local variable. Any value read in from the database is copied into local variables with the same names as the corresponding database items. The database items are only changed with a write statement. If initially A: {400, W_a, R_a} and B: {500, W_b, R_b}, indicate their values after steps 3, 5, 7, 8, 12 and 14.

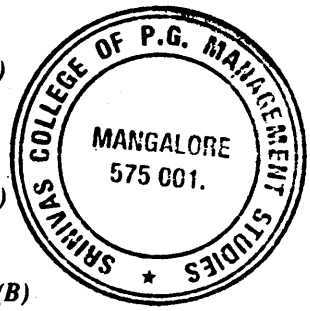
Figure M Schedule for Exercise 12.8.

Step	Schedule	Transaction T_{22}	Transaction T_{23}
1	*Sum* := 0	*Sum* := 0	
2	*Sum* := 0		*Sum* := 0
3	**Read**(*A*)		**Read**(*A*)
4	*A* := *A* − 100		*A* := *A* − 100
5	**Read**(*A*)	**Read**(*A*)	
6	*Sum* := *Sum* + *A*	*Sum* := *Sum* + *A*	
7	**Write**(*A*)		**Write**(*A*)
8	**Read**(*B*)	**Read**(*B*)	
9	*Sum* := *Sum* + *B*	*Sum* := *Sum* + *B*	
10	**Show**(*Sum*)	**Show**(*Sum*)	
11	*Sum* := *Sum* + *A*		*Sum* := *Sum* + *A*
12	**Read**(*B*)		**Read**(*B*)
13	*B* := *B* + 100		*B* := *B* + 100
14	**Write**(*B*)		**Write**(*B*)
15	*Sum* := *Sum* + *B*		*Sum* := *Sum* + *B*
16	**Show**(*Sum*)		**Show**(*Sum*)

12.9 We have three transactions, T_{24}, T_{25}, and T_{26}, with timestamp values of t_{24}, t_{25}, and t_{26}, respectively ($t_{24} < t_{25} < t_{26}$). The schedule for the concurrent execution of these transactions is given in Figure N. Assuming that initially A: a, W_a, R_a and B: b, W_b, R_b and C: c, W_c, R_c, show these values after each step if the timestamp-ordering scheme for concurrency control is used.

Figure N Schedule for Exercise 12.9.

Step	Schedule	Transaction T_{24}	Transaction T_{25}	Transaction T_{26}
1	**Read**(A)	**Read**(A)		
2	$A := f_1(A)$	$A := f_1(A)$		
3	**Read**(B)			**Read**(B)
4	**Write**(A)	**Write**(A)		
5	**Read**(C)		**Read**(C)	
6	$C := f_2(C)$		$C := f_2(C)$	
7	**Read**(C)			**Read**(C)
8	**Write**(C)		**Write**(C)	
9	**Read**(B)	**Read**(B)		
10	$B := f_3(B)$			$B := f_3(B)$
11	**Write**(B)			**Write**(B)
12	$B := f_4(B)$	$B := f_4(B)$		
13	**Write**(B)	**Write**(B)		



12.10 Suppose we want to add a record occurrence to record type R_1, (Figure 12.19) which uses indexes I_{11} and I_{12} for direct access to the records. Give the sequence of locking to perform this operation.

12.11 Algorithm 12.2 is inefficient because some transactions are processed many times. Give a modification to the algorithm to avoid this inefficiency.

12.12 In an adaptive deadlock detection scheme, why is it necessary to choose an upper and lower limit for the frequency of running the deadlock detection algorithm?

12.13 In the concurrency control scheme based on timestamp ordering, we have assumed that the timestamp value is based on a systemwide clock. Instead of using such a timestamp to determine the ordering, suppose a pseudorandom number generator was used. Show how you would modify the concept of older and younger transactions with this modification and give the modified wait-die and wound-wait protocols.

Bibliographic Notes

Gray in (Gray 79) presents comprehensive operating system requirements for a database system. The transaction concept and its limitations are discussed in (Gray 81). The serializability concept, the two-phase locking protocol, and its correctness is due to the early work by Eswaran et al. (Eswa 79) in connection with System R. The extension of the serializability test for read-only and write-only cases are discussed in (Papa 79). The algorithm for this case is developed in (Bern 79), and the text by Ullmann (Ullm 82) also treats this topic. Locking schemes, multigranularity, and intention-locking extensions are discussed in (Gray 75). Extensions to lock modes and deadlock avoidance are discussed in (Kort 82) and (Kort 83).

(Reed 79) presented the earliest known multiversion timestamping algorithm. The use of a pseudotimestamp was discussed in (Reed 83) and (Svob 80). It is shown in (Bern 83) that any schedule generated according to the timestamp concurrency control algorithm requirements is serializable, and the result obtained by a set of concurrent transactions is the same as obtained by some serial execution of the set of transactions with a single version of the data-

items. The reader interested in the multiversion concurrency control algorithms based on locking is referred to (Baye 80) and (Ster 81). The extension of the locking scheme and locking with timestamp ordering (combination scheme) is discussed in (Bern 83). The combination scheme was discussed in (Chan 82). The tree-locking protocol for a database whose storage is tree structured is discussed in (Silb 80) and this protocol is generalized to the read-only and write-only locks in (Kade 80). (Bern 80) presents a number of different distributed database concurrency control schemes based on timestamping.

An optimistic method for concurrency control is presented in (Kung 81). (Rose 79) proposed the wait-die and wound-wait transaction retry schemes to avoid deadlocks in a distributed database system, although these schemes are applicable to a centralized database system as well.

The deadlock problem is surveyed in (Coff 71) and (Holt 72). (Islo 80) discusses the general deadlock problem and examines the problems unique to database systems, both centralized and distributed.

Bibliography

(Bass 88) M. A. Bassiouni, "Single-Site and Distributed Optimistic Protocols for Concurrency Control," *IEEE-SE* SE 14 (8), August 1988, pp. 1071–1080.

(Baye 80) H. Bayer, H. Heller, & A. Reiser, "Parallelism and Recovery in Database Systems," *ACM TODS* 5(4), June 1980, pp. 139–156.

(Bern 79) P. A. Bernstein, D. W. Shipman, & W. S. Wong, "Formal Aspects of Serializability in Database Concurrency Control," *IEEE-SE* SE 5 (3), May 1979, pp. 203–215.

(Bern 80) P. A. Bernstein, & N. Goodman, "Timestamp-Based Algorithms for Concurrency Control in Distributed Systems," Proc. 6th International Conf. on Very Large Data Bases, Montreal, October 1980, pp. 285–300.

(Bern 83) P. A. Bernstein, & N. Goodman, "Multiversion Concurrency Control—Theory and Algorithms," *ACM TODS* 8(4), Dec. 1983, pp. 465–483.

(Caso 81) M. A. Casonova, "The Concurrency Control Problem of Database Systems," *Lecture Notes in Computer Science*, vol. 116. New York: Springer-Verlag, 1981.

(Chan 82) A. Chan, S. Fox. W. T. K. Lin, A. Nori, & D. R. Ries, "The Implementation of an Integrated Concurrency Control and Recovery Scheme," Proc. ACM/SIGMOD Conf. on Management of Data, Orlando, Florida, June 1982, pp. 184–191.

(Coff 71) E. G. Coffman, M. J. Elphick, & A. Shoshani, "System Deadlocks," *ACM Computing Surveys* 3(2), June 1971, pp. 67–88.

(Eswa 79) K. P. Eswaran, J. N. Gray, R. A. Lorie, & I. L. Traiger, "The Notion of Consistency and Predicate Locks in a Database System," *CACM*, 19(11), November 1979, pp. 624–633.

(Gray 75) J. N. Gray, R. A. Lorie, & G. R. Putzolu, "Granularity of Locks in a Shared Data Base," Proc. of the VLDB, 1975, pp. 428–451.

(Gray 79) J. N. Gray, "Notes on Data Base Operating Systems," in R. Bayer, R. M. Graham, & G. Seegmuller, eds., *Operating Systems: An Advanced Course*. Berlin: Springer-Verlag, 1979.

(Gray 81) J. N. Gray, "The Transaction Concept: Virtues and Limitations," Proc of the 7th VLDB Conference, 1981, pp. 144–154.

(Holt 72) R. C. Holt, "Some Deadlock Properties of Computer Systems," *ACM Computing Surveys* 4(3), September 1972, pp. 179–196.

(Hunt 79) H. B. Hunt, & D. J. Rosenkrantz, "The Complexity of Testing Predicate Locks," Proc. ACM-SIGMOD 1979 International Conference on Management of Data, May 1979, pp. 127–133.

(Islo 80) S. S. Isloor, & T. A. Marsland, "The Deadlock Problem: An Overview," *Computer* 13(9), September 1980, pp. 58–78.

(Kade 80) Z. Kadem, & A. Silberschatz, "Non-Two Phase Locking Protocols with Shared and Exclusive Locks," Proc. 6th International Conf. on Very Large Data Bases, Montreal, October 1980, pp. 309–320.

(Kort 82) H. F. Korth, "Deadlock Freedom Using Edge Locks," *ACM TODS* 7(4), December 1982, pp. 632–652.

(Kort 83) H. F. Korth, "Locking Primitives in a Database System," *ACM JACM* 30(1), January 1983, pp. 55–79.

(Kung 79) H. T. Kung, & C. H. Papadimitriou, "An Optimality Theory of Concurrency Control for Databases," Proc. ACM-SIGMOD 1979 International Conference on Management of Data, May 1979, pp. 116–126.

(Kung 81) H. T. Kung, & J. T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Systems* 6(2), June 1981, pp. 213–226.

(Lync 83) N. A. Lynch, "Multilevel Atomicity—A New Correctness Criterion for Database Concurrency Control," *ACM TODS* 8(4), September 1983, pp. 484–502.

(Papa 79) C. H. Papadimitriou, "The Serializability of Concurrent Database Updates," *JACM* 26(4), October 1979, pp. 150–157.

(Reed 79) D. P. Reed, "Naming and Synchronization in a Decentralized Computer System," MIT/LCS/TR-205, Cambridge, MA: MIT, September 1979.

(Reed 83) D. P. Reed, "Implementing Atomic Actions on Decentralized Data," *ACM Transactions on Computer Systems* 1(1), pp. 3–23.

(Rose 79) D. J. Rosenkrantz, R. E. Stearns, & P. M. Lewis II, "System Level Concurrency Control for Distributed Data Base Systems," *ACM TODS* 3(2), March 1978, pp. 178–198.

(Silb 80) A. Silberschatz, & Z. Kadem, "Consistency in Hierarchical Database Systems," *JACM*, 27(1), January 1980, pp. 72–80.

(Ster 81) R. E. Stern, & D. J. Rosenkrantz, "Distributed Database Concurrency Controls Using Before-Values," Proc. ACM/SIGMOD Conf. on Management of Data, 1981, pp. 74–83.

(Svob 80) L. Svobodova, "Management of Object Histories in the Swallow Repository," MIT/LCS/TR-243, Cambridge, MA: MIT, July, 1980.

(Ullm 82) J. D. Ullman, *Principles of Database Systems*. Rockville, MD: Computer Science Press, 1982.

- **Database security:** Protection of the information contained in the database against unauthorized access, modification, or destruction.
- **Database integrity:** The mechanism that is applied to ensure that the data in the database is correct and consistent. The term **semantic integrity** is sometimes used to refer to the need for maintaining database consistency in the presence of user modifications. Semantic integrity is maintained either implicitly by the data model, or is specified explicitly by appropriate constraints on the data that can be entered by the user, and this data is checked by the DBMS. Entity and referential integrity constraints are implicit in the relational data model. Set insertion and retention rules are implicit in the network model. A record occurrence in the network model is restricted to be a member in only one occurrence of a set type. The requirement that an instance of a child type record cannot exist without the parent record occurrence is implicit in the hierarchical model. We discuss integrity issues further in Section 13.4.
- **Authorization:** The culmination of the administrative policies of the organization, expressed as a set of rules that can be used to determine which user has what type of access to which portion of the database. Persons who are in charge of specifying the authorization of different portions of the database are usually called **security administrators** or **authorizers.**

13.2 Security and Integrity Threats

Some types of threats can only be addressed using social, behavioral, and control mechanisms such as ethical training, expected conduct by the employees of an organization, and appropriate legislation. These threats include actions on the part of authorized users to perform actions such as deliberately adding unauthorized users, giving some users more access than required for their normal operations, divulging passwords, and threatening bribery and blackmail. However, in spite of the most stringent legislation and penalties for transgressions, there will always be lapses in any system, computerized or not. The intention of the DBMS is to make it unprofitable, economically or otherwise, for casual users to breach the security mechanism.

In addition to features required in the DBMS for security and integrity, additional requirements have to be supported by the operating system and the protocol for physical access to the computing system itself.

The operating system must ensure that files belonging to the database are not used directly without proper authorization. This authorization can consist of the user providing the proper passwords for the file. The operating system must also ensure that illegal users using public communication facilities are not allowed access to the system. Users must be required to use adequate identification and passwords (passwords must be sufficiently long and must be changed frequently to thwart intruders and hackers).

Access to the computing facility and the storage medium must be restricted to authorized persons only. There must be adequate physical protection, as in the case of any valuable asset. Disposal of old storage devices must be done in a proper manner. Any sensitive data resident on storage devices to be disposed of must be destroyed.

In a telecommunications environment, data may be accessed by eavesdroppers, wiretappers, and other illegal users. To prevent this type of threat, data transmitted over public communication channels should be in a ciphered form.

We can classify security and integrity threats in the categories of accidental, or intentional or malicious.

Accidental Security and Integrity Threats

- A user can get access to a portion of the database not normally accessible to that user due to a system error or an error on the part of another user. For example, if an application programmer accidentally omits appropriate verification routines, the resulting programs would compromise the database.
- Failures of various forms during normal operation, for example, transaction processing or storage media loss. Proper recovery procedures are normally used to recover from failures occurring during transaction processing. Lack of such procedures could lead to inconsistencies in the database as discussed in Chapter 11.
- Concurrent usage anomalies. Proper synchronization mechanisms are used to avoid data inconsistencies due to concurrent usage. We discussed these problems in Chapter 12.
- System error. A dial-in user may be assigned the identity of another dial-in user who was disconnected accidentally or who hung up without going through a log-off procedure.
- Improper authorization. The authorizer can accidentally give improper authorization to a user, which could lead to database security and/or integrity violations.
- Hardware failures. For example, memory protection hardware that fails could lead to software errors and culminate in database security and/or integrity violations.

Malicious or Intentional Security and Integrity Threats

- A computer system operator or system programmer can intentionally bypass the normal security and integrity mechanisms, alter or destroy the data in the database, or make unauthorized copies of sensitive data.
- An unauthorized user can get access to a secure terminal or to the password of an authorized user and compromise the database. Such users could also destroy the database files.
- Authorized users could pass on sensitive information under duress or for personal gain.
- System and application programmers could bypass normal security in their programs by directly accessing database files and making changes and copies for illegal use.
- An unauthorized person could get access to the computer system, physically or by using a communications channel, and compromise the database.

and the structure of the data has to be determined by the DBA (who could also be designated as owner). Procedures for modifications to the security control mechanism must also be enacted.

Access Control Policies

In addition to the administrative procedures, the lower level access control policies have to be determined in light of the security features provided by the DBMS and OS. Access control policies can be classified as follows:

- **Open vs. closed system:** In an open system, a user is allowed access to everything unless access is explicitly denied. In a closed system, a user is not allowed to access anything unless access is explicitly granted. A closed system enforces the **least privilege** or the **need-to-know** policy; an open system **maximizes sharing** of information and minimizes the portion that is not to be known.
- **Content-independent access control:** This policy is also called **name-dependent access control.** Access is allowed to those data objects whose names are known to the user. A data object can be a relation name and some of the associated attributes in the case of a relational database. In the case of a network database, it could be a set with the owner and member record types, with some of the associated data fields. Thus, access is independent of the contents of the data object. Consider the relation of Figure 13.1. All the employees in an organization may have content-independent access to the data object EMPLOYEE*(Employee_Name, Department, Room, Phone_No)*. The manager of the Personnel department, however, has content-independent access to the entire data object EMPLOYEE *(Employee_Name, Department, Room, Phone_No, Position, Salary)*.
- **Content-dependent access control:** In this policy the concept of least privilege can be extended to take into account the contents of the database and result in finer granularity of access control. The chairperson of a department can have content-independent access to EMPLOYEE*(Employee_Name, Department, Room, Phone_No)* and content-dependent access to EMPLOYEE*(Employee_*

Figure 13.1 The EMPLOYEE relation.

Employee_Name	*Department*	*Room*	*Phone_No*	*Position*	*Salary*
Smith	Comp Sci	A632	848-3876	Asst Prof	44500
Clark	Comp Sci	A651	848-3874	Asso Prof	49750
Turner	Chemistry	C643	848-2981	Professor	63050
Jamieson	Mathematics	M728	848-3829	Professor	61430
Bosky	Physics	P388	848-1286	Asso Prof	52800
Newton	Physics	P391	848-1291	Asst Prof	42750
Mann	Elect Eng	E389	848-8628	Asst Prof	44750

Figure 13.2 The HEAD relation.

Chairperson	*Secretary*	*Department*
Smith	Rolland	Comp Sci
Jamieson	Evans	Mathematics
Bosky	Fuhr	Physics
Turner	Horngren	Chemistry
Mann	Messer	Elect Eng

Name, Department, Room, Phone_No, Position, Salary), such that the EMPLOYEE.*Department* is the department where she is the chairperson. This can be implemented by a query modification as shown below:

select *(Employee_Name, Salary)*
from EMPLOYEE
where *Department* = Comp Sci

The above query can be modified as shown below, assuming that there is a relation HEAD with attributes *(Chairperson, Secretary,Department)* as shown in Figure 13.2.

select *(Employee_Name, Salary)*
from EMPLOYEE
where *Department* = (**select** *(Department)*
from HEAD
where *Chairperson* = user's name)

Access Operation Type Control Policies

Greater control over the use of data is obtained when the security policy distinguishes the type of access that is allowed to a data object. The classification of access to a data object known to the user can be as follows: read, update, insert, delete. Thus, everyone in an organization may be allowed access to the data object EMPLOYEE *(Employee_Name, Department, Room, Phone_No)* with the access type being read. The departmental secretary may be assigned update access to the EMPLOYEE.*Room* and EMPLOYEE.*Phone_No* data items, and this update access may be content dependent only to occurrences of the secretary's department. This can be implemented by a query modification as follows:

update EMPLOYEE
Room = new room
Phone_No = new phone number
where *Employee_Name* = somename

The above query may be modified as follows to ensure that the departmental secretary modifies only the tuples for his own department's employees:

tains rows called **subjects** and columns termed **objects.** The entr
the position corresponding to the intersection of a row and colum
of access that the subject has with respect to the object.

Objects

An object is something that needs protection and one of the first steps in the authorization process is to select the objects to be used for security enforcement. A typical object in a database environment could be a unit of data that needs to be protected. However, the unit of data could be at some convenient size or granularity. Thus, a data field, a record, or a file could be considered an object. Another type of object that can be protected is a view or subscheme. Using views as objects and hence as units of protection automatically limits the amount of the database that can be accessed by a user.

The objects in the access matrix represent content-independent access control. However, to enforce content-dependent access control, some structure for conditions or access predicates are incorporated in the access matrix. Some examples of access predicates, expressed as query modifications, are shown in Figure 13.4.

Views as Objects

In addition to providing different ways of looking at the data in the database, views or subschemes can be used to enforce security. A user is allowed access to only that portion of the database defined by the user's view. A number of users may share a view. However, the user may create new views based on the views allowed. The advantage of this approach is that the number of objects accessible to a class of users and the entry for it in the authorization matrix is reduced to one per view. This reduces the size of the authorization matrix. The disadvantage is that the entire class of users have the same access rights.

Granularity

The usual practice is to choose the granularity of security enforcement. This could be a file, a record (relation), or a data item (attribute). The smaller the protected object, the finer the degree of specifying protection. However, the finer granularity increases the size of the authorization matrix and the overhead in enforcing database security.

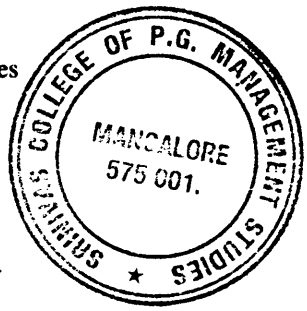
Subject

A subject is an active element in the security mechanism; it operates on objects. A subject is a user who is given some rights to access a data object. We can also treat a class of users or an application program as a subject. A user who belongs to or joins a class of users gets the access rights of that class of users. If a user belongs to more than one class of users, then the access rights for a given access made by the user depends on the class of user that is being used by that user for the access.

Access Types

The access allowed to a user could be for data manipulation or control. The manipulation operations are read, insert, delete, update. The control operations are add, drop, alter, and propagate access control. We define these operations below:

- **Read:** Allows reading only of the object.
- **Insert:** Allows inserting new occurrences of the object type, for example, a tuple in a relation. Insert access type requires that the subject has a read access as well. However, an insert access may not allow modification of existing data.
- **Delete:** Allows deleting an existing occurrence of the object type.
- **Update:** Allows the subject to change the value of the occurrence of the object. Some data-items in a record, such as the primary key attributes, however, may not be modified. For reasons discussed in Section 5.4.1, update through a view may or may not be allowed. An update authorization may not include a delete authorization as well.
- **Add:** Allows the subject to add new object types such as new relations (in relational systems), record and set types (in network systems), or record types and hierarchies (in hierarchical systems).
- **Drop:** Allows the subject to drop or delete existing object types from the database. Here we are referring to the deletion of a type and not of an occurrence.
- **Alter:** Allows the subject to add new data-items or attributes to an existing record type or relation; also allows the subject to drop existing data-items or attributes from existing record types or relations.
- **Propagate access control:** This is an additional right that determines if this subject is allowed to propagate the right over the object to other subjects. Thus, a subject S may be assigned an access right R over an object O, and in addition the right to grant this access right (or part of it) to another subject.



In the access matrix of Figure 13.4, we have indicated both content-independent access rights and content-dependent access rights; the latter have been indicated with query modification clauses.

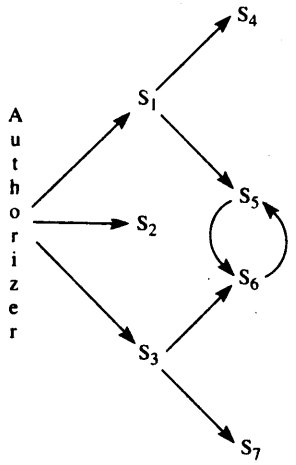
In addition to the above access rights, a subject may have the privilege to create additional indexes for a record type or relation, execute certain application programs (another type of object), and so on.

Authorization Grant Tree

Consider a user subject. When the user has the propagate access control right over an object, he or she can pass all or part of her or his right to another subject, for instance another user. In a organization that uses the centralized security administration policy, the authorizer has all the access rights including the propagate access control right over the database. When the authorizer grants a user some rights this may be granted with the propagate access control as well. This leads to an **authorization grant tree**, as shown in Figure 13.5.

To properly revoke access rights, all paths in the access grant tree must start from the authorizer, otherwise the revocation cannot be guarded from unscrupulous

Figure 13.5 Authorization grant tree.



usage. With this proviso, revocation of the access rights of subject S_3 in Figure 13.5 means that subject S_7 also loses all rights. Subject S_6 retains those rights granted by S_5 and S_5 loses rights granted by S_6. We illustrate this pruned access grant tree in Figure 13.6.

Further revocation of access right of subject S_5 by S_1 causes S_6 to lose all access rights as well; this is illustrated in Figure 13.7.

Without the requirement that a direct path exists from the authorizer, the reader can verify that S_5, S_6, and S_7 would retain their access rights when the authorizer revokes the access right of subject S_3, as illustrated in Figure 13.8.

Figure 13.6 Authorization grant tree after revocation of access rights from S_3; S_6 retains the access right granted by S_5.

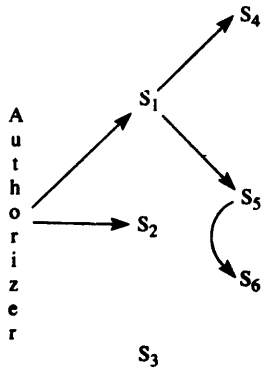
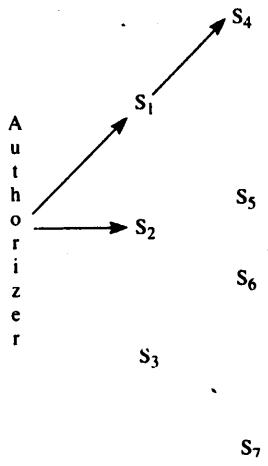


Figure 13.7 Authorization grant tree after revocation of access rights from S_5; S_6 also loses access rights.

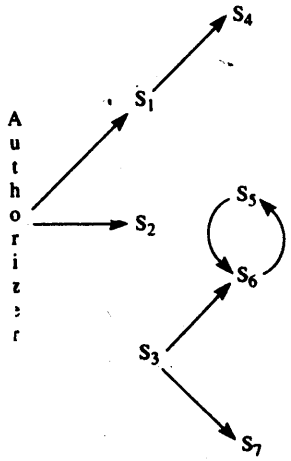


Authorization Facilities

The facility available to the authorizer (and to the users who can propagate access rights) to assign access rights could be in the form of a separate language or could be integrated with the data definition or the data manipulation language.

In the network model the access rights specifications are integrated with the data definition language. The subschema can be used to grant access to a subset of the database to an user. However, it does not provide a facility to indicate the operations that a user can perform on the portion of the database accessible to the user.

Figure 13.8 Authorization grant tree with access rights that cannot be properly revoked.



be designed to require the user to provide a password before allowing a sensitive operation.

Instead of a simple password, the system may ask the user one or more questions from a set of questions; only the user can correctly answer these questions. One such scheme involves generating a pseudorandom number X and prompting the user to respond with $T(X)$, where T is a prearranged simple transformation function. Since only the user and the system know what this prearranged transformation T is, anyone eavesdropping will only see X and $T(X)$ and cannot easily discern T. Each authorized user in this method of authentication is supplied with unique transformation function.

Something in the User's Possession

In this scheme, each user could be given an appropriately encoded badge, card, or key to be used for identification purposes. A password or question-answering scheme as before can be used for the authentication purpose.

Some Characteristic of the User

In this scheme, the identification and authentication procedures are combined in one step, but require the use of special hardware and software to identify some physical or physiological characteristic of the user. These characteristics are known to be unique or have a very low probability of duplication in a population of a given size, and hence cannot be easily faked. Examples of such characteristics are fingerprints or the relative lengths of the fingers of a hand. Another scheme that has been proposed is the use of voiceprint; however, a simple technique like using a tape recording of the authorized user's voice can be used to impersonate the user.

13.3.4 Views/Subschemes in Security Enforcement

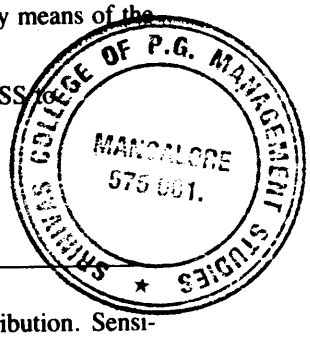
The content of the database is described by the conceptual scheme and the users' views are defined by the subschemes. The subscheme can be used in the name-dependent security enforcement policy to limit the portion of the database known to and hence accessible by a user. The network model as proposed in the DBTG uses the subschema as the major security enforcement mechanism. A user is not allowed access to anything that is not included in that user's subschema.

The following example illustrate creating a view for use by the departmental secretary consisting of the attributes *Employee_Name*, *Room*, and *Phone_No*. The tuples accessible are limited to the employees in the secretary's department.

```
create view EMP_ADDRESS (Name,Room_No, Phone) as
(select (e.Employee_Name,e.Room, e.Phone_No)
 from EMPLOYEE e
 where e.Department = (select (Department)
                       from HEAD
                       where Secretary = 'secretary_name'))
```

Having created this view, the secretary is granted appropriate access rights to any tuple of this relation and is allowed to update *Room_No, Phone* by means of the following grant statement:

grant select update *(Room_No, Phone)* **on table** EMP_ADDRESS **to**
'secretary_name'



13.3.5 Distributed Systems

Security enforcement in distributed systems can be enhanced by distribution. Sensitive information can be fragmented and stored at dispersed sites. The leakage of some portion of the fragmented data may be not as disastrous as the leakage of unfragmented data. Also, with distribution, different sites can have different levels of security. However, in this case, the more secure sites have to take into account the existence of less secure sites in transmitting data over the network. Since data will be transmitted over a communication channel, appropriate encryption schemes (discussed in Section 13.3.6) should be used.

The authorization functions in a distributed system have to be decentralized and a decision has to be made as to where to store the access matrix or access rules. One possible choice is to fragment the access matrix and store the appropriate fragments at the sites of the data fragments.

13.3.6 Cryptography and Encryption

Consider the secure transmission of this message:

"Mr. Watson, can you please come here."

One method of transmitting this message is to substitute a different character of the alphabet for each character in the message. If we ignore the space between words and the punctuation, and if the substitution is made by shifting each character by a different random amount, then the above message can be transformed into, e.g., the following string of characters:

"xhlkunsikevoabondwinhwoajahf."

Cryptography has been practiced since the days of the Roman Empire. With the increasing use of public communication facilities to transmit data, there is an increased need to make such transmissions secure. In a distributed environment, transmitting highly confidential information between geographically dispersed sites, in spite of the most stringent local security enforcement, could lead to leakage from eavesdropping and wiretapping.

This points to the need for the data to be encrypted before it is transmitted. At the receiving end, the received data is deciphered before it is used. The sender must know how to encrypt the data and the receiver must know how to decipher the coded message. Since the computers at both ends can be used to cipher and decipher the data, the code used for ciphering can be quite complex.

In this section we consider some types of constraints that the database has to enforce to maintain the consistency and validity of data. One aspect that has to be dealt with by the integrity subsystem is to ensure that only valid values can be assigned to each data-item. This is referred to as **domain integrity.** Another set of integrity constraints are the so-called structural and semantic constraints. Some of these types of constraints are addressed by the data models used and others are addressed in the design of the database by combining appropriate functional dependencies in different records. Some if not most of the functional dependencies can be expressed if the DBMS allows each record type or relation to have an associated primary key. We discuss these aspects below.

In traditional systems, application programs were responsible for the validation of data and maintaining the consistency of the data used by the programs. However, in a DBMS environment, depending on the application programs to perform these checks has the following drawbacks:

- Each application program must have correct validation and consistency check routines; a failure in one program could lead to database inconsistencies.
- Each application program must be aware of the semantics of the complete database to enforce the correct consistency checks; this is not always the case and unnecessarily burdens the application program writers.
- There will be considerable duplication of efforts.
- Integrity constraints are hard to understand when they are buried in the code of application programs.
- No consistency or validity checks are possible for direct database manipulation using a query language.

Centralizing the integrity checking directly under the DBMS reduces duplication and ensures the consistency and validity of the database. The centralized integrity constraints can be maintained in a system catalog (data dictionary) and can be accessible to the database users via the query language. This does not rule out an application program performing some specific checking, including input validation.

13.4.1 Domain or Data-Item Value Integrity Rules

One of the most common integrity constraints that is specified and validated is to define the domain for each attribute, or in the case of network or hierarchical models, to define the value set for each data-item. Domain integrity rules are simply the definition of the domains of the attributes or the value set for the data-items. The value that each attribute or data-item can be assigned is expressed as being one or more of the following forms: a data type, e.g., alphanumeric string or numeric; a range of values; or a value from a specified set. For instance, in the relation EMPLOYEE of Figure 13.1, the domain of the attribute *Salary* may be given as being between $12,000 and $300,000. The final *Grade* assigned to a student in a course can only be one of, say, A, B, C, D, or F.

A domain can be composite; for instance, the *Date* attribute in the relation MED_HISTORY is restricted to the form mm/dd/year, where mm is the month and is restricted to the range 01 through 12; dd is the date and is restricted to the range

01 through 31; and year is, say, 1986 through 2000. We can make the range of dd more precise by taking into account both the month and year.

Since online data entry is a common operation, validation of the entered values has to be performed to maintain the integrity of data. Traditionally the validation was performed by application programs. However, this approach has two drawbacks: first, it depends on the application programmer to include all validity checks, and second, each application program is duplicating some of these checks. Hence, it is preferable to centralize these operations and let the DBMS perform the validity checks. Note that some types of errors cannot be detected. For instance, a professor may incorrectly assign a grade of F instead of D to a student (an accidental error perhaps, because the keys for D and F are next to each other on the QWERTY keyboard). The validation procedure cannot detect this as an error, since F is a valid grade. Thus, integrity mechanisms can only ensure that the data is in the specified domain. Incorrect choices, as long as they do not violate any integrity constraints, are not considered to be errors.

Some domain constraints could be conditional. For example, the salary constraint in the EMPLOYEE relation, instead of being restricted to a given range could be restricted conditionally as follows:

if *Position* is Asst. Prof *Salary* must be between 35,000 and 45,000
if *Position* is Asso. Prof *Salary* must be between 42,000 and 55,000
if *Position* is Professor *Salary* must be between 53,000 and 200,000

The domain values supplied for an operation are validated against the domain constraints. Any violation of a domain integrity rule typically results in the operation being rejected with an appropriate message returned to the user for the correct value. Other possible choices of action to be undertaken by the DBMS on the detection of a domain constraint violation are: correct the value to a valid value; replace the value with a sentinel value that will be detected at audit time; roll back the transaction that issued the invalid value.

The validation procedure typically runs after each attempted modification; however, some integrity constraints may be validated only after the completion of a transaction. Consider the total quantities of some part in a plant. This value must not change unless there is a shipment or receipt of that part. If a transaction transfers 100 units of the part from inventory to a project in the plant, the total units of that part will be incorrect after the first operation of the transaction, which subtracts 100 units from the quantity on hand in inventory, and before the end of the second operation of the transaction, which adds 100 units to a project. The database is in an inconsistent state if the total for the part being transferred were to be computed after the first operation was completed.

In specifying the domain constraints, null values may or may not be allowed Thus, it is usual not to allow null values for any attribute that forms part of a primary key of a relation.

The definition of the EMPLOYEE relation of Figure 13.1 can be given as shown below, where some of the domain constraints are included. The attribute *Employee_Name* is declared as a primary key that must not be null.

type EMPLOYEE = *relation*
Employee_Name alphabetic string length 25 *unique null not allowed*
Department alphabetic string length 15 *values* (CompSci, Chemistry, Electrical Engineering, Mathematics, Physics, . . .)

13.4.3 Violation of Integrity Constraints and Corrective Action

As mentioned earlier, the validation of the database can be done right after the completion of a single request to the database; at some point within a transaction, including at the end of a transaction; or at some time specified by the DBA or a database auditor (the latter may be called the **audit time**).

If the validation is done after each request to the database, a message can be returned to the user or application program indicating the problem, and the request will fail. If the validation checks are performed at some point within a transaction (including just before it is committed), there is a requirement to perform a maintenance operation in case of integrity violation. This would involve terminating the transaction and undoing any changes made by the transaction.

If the validation checks are done at audit time, it becomes difficult to assign the integrity violation to a single database request or a single transaction. An audit trail could be helpful in pinpointing the culprit; however, corrective actions have to be performed on transactions that were processed from the time of the integrity violation.

13.4.4 A General Model of Integrity

A general integrity constraint can be specified using a model that gives the following parameter for each constraint:

- **D:** The data object(s) to which the constraint applies.
- **O:** The database operation for which the constraint will be tested.
- **A:** The assertion or semantic constraint that must be satisfied by the occurrence of the data object(s).
- **C:** Predicates to be applied to the data object. The predicates select those occurrences of the data object to which the assertion **A** will be applied. If the condition holds for a given occurrence of **D**, it is a candidate for the constraint **A**.
- **P:** The procedure (sometimes called an **auxiliary procedure**) that will be triggered for execution when an integrity violation is found to be true (if the condition **A** is not true). The auxiliary procedure must take corrective action to maintain integrity.

Using this model, each constraint can be expressed as a five-tuple: **(D, O, A, C, P)**.

The auxiliary procedure **P** in the above model is said to be triggered when a modification to the database causes an integrity violation, i.e., the constraint **A** does not hold. The procedure is responsible for taking corrective actions.

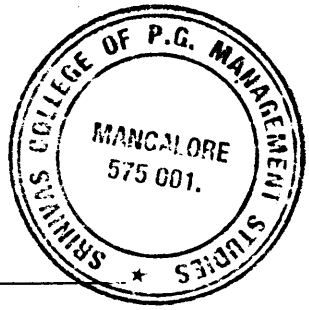
One type of operation that the auxiliary procedure can be called to take is to check some complex integrity requirements that cannot be specified by assertions. A method of triggering such a procedure would be by setting the assertion **A** to false and the condition **C** to true in the constraint rule. In addition, such an auxiliary procedure could be called to prepare appropriate audit trails, and so forth.

It has been proposed that an integrity mechanism called **trigger** be included in the new standard for SQL. A trigger is defined as follows:

define trigger trigger_name
on relation names
predicate(s)
action auxiliary procedure

We give below an example of trigger definition where the salary of employees is checked on insertion or update:

define trigger *salary_validation*
on relation EMPLOYEE
EMPLOYEE.*Salary* > 10000 **and** Employee.*Salary* < 200000
action Notify_Personnel_Manager



13.4.5 Expressing Integrity Constraints

Most DBMSs have some form of language constructs for expressing domain and key constraints. These constructs could be part of the data definition language, the data manipulation language, or a special language. However, the constructs for expressing complex constraints are only in the evolution stage. We gave a form for checking a general constraint in Section 13.4.4 and the **define trigger** statement proposed for SQL.

Since the DBMS is representing a given data model, it is aware of some of the integrity constraints implicitly built into the data model. It is informed of the record structure and other implicit integrity constraints by the declaration in the data definition language. For instance, in a hierarchical system the declaration of a record type gives its structure; in addition, some data fields may be declared as unique to specify a primary key of the record type. The hierarchies with the parent and dependent record type gives the relationship between record types. Additional rules, as mentioned in Chapter 9, may be specified which could result in the creation of logical or virtual parents and enforce appropriate referential integrity constraints and semantic consistencies.

The network data definition facility allows the definition of a primary key (by not allowing duplicates). The check clause in the data definition can be associated with each data-item that specifies the valid values or data type. The insertion and retention rules for sets define the semantics and referential integrity of the independent existence of the member record type occurrence vis-à-vis the owner record occurrence. The **check** clause is also used to specify other arbitrary constraints, and it may be formulated to enforce constraints between distinct record types, stipulating operations that will trigger the execution of an associated auxiliary procedure.

Relational data definition language also provides statements to allow specification of constraints. The **assert** statement is one such statement. The assert indicates that a constraint is specified involving relations in the **on clause.** The assertion to be enforced is given by predicates following the list of relations. However, current relational languages and DBMSs support such a statement only partially.

integrity constraints should be enforced for all update operations. Finally, appropriate audit trails should be generated.

13.7 Summary

Security and integrity concepts are crucial since modifications in a database require the replacement of the old values, but the fact that there was an old value for a given data item is not evident. The DBMS security mechanism restricts users to only those pieces of data that are required for the functions they perform. Security mechanisms restrict the type of actions that these users can perform on the data that is accessible to them. The data must be protected from accidental or intentional (malicious) corruption or destruction. In addition there is a privacy dimension to data security and integrity.

Four levels of defense are generally recognized for database security: human factors, administrative controls, physical security, and the security and integrity mechanisms built into the operating system and the DBMS. Access control policies are classified as open vs. closed systems, content-independent access control, content-dependent access control, access operation type control, access context control, access control based on history of accesses, and information flow policies. The database depends on protection mechanisms such as user identification and validation as well as the memory and file protection features of the OS.

Authorization is the outcome of the administrative policies and is expressed as a set of rules that can be used to determine which user has what type of access to which portion of the database. The person who is in charge of specifying the authorization is called the authorizer. The authorization is usually maintained in the form of an access matrix, containing rows called the subjects and columns called the objects. An object is something that needs protection. The entry in the matrix at the position corresponding to the intersection of a row and column indicate the type of access that the subject has with respect to the object. Views or subschemes can be used to enforce security. A user is allowed access only to that portion of the database that is defined by the user's view. A user may be granted some rights with the propagate access control, which leads to the existence of an authorization grant tree. To revoke access rights, all paths in the grant tree must start from the authorizer, otherwise the revocation cannot be guarded against unscrupulous usage.

The facility available to the authorizer (and to the users who can propagate access rights) to assign access rights could be in the form of a separate language or could be integrated with the data definition or data manipulation language.

The user has to identify herself/himself to the system and authenticate the identification. Security enforcement in distributed systems can be enhanced by distribution; thus, sensitive information can be fragmented and stored at dispersed sites.

With the increasing use of public communication facilities to transmit information, there is a need for the data to be encrypted before it is transmitted and this requires that, at the receiving end, the received data be deciphered. A public key encryption scheme can be used. In this scheme both the encryption key and the encryption algorithm are public and readily available. However, the decryption key is secret; only the rightful recipient can decipher the coded message.

Security constraints guard against accidental or malicious tampering with data;

integrity constraints ensure that any properly authorized access, alteration, deletion, or insertion of the data in the database does not change the consistency and validity of the data. Database integrity involves the correctness of data and this correctness has to be preserved in the presence of concurrent operations, errors in the user's operations and application programs, and failures in hardware and software. One aspect to be dealt with by the integrity subsystem is to ensure that only valid values can be assigned to each data-item; this is referred to as domain integrity. Another set of integrity constraints are the so-called structural and semantic constraints. Some of these types of constraints are addressed by the data models and others are addressed in the design of the database by combining appropriate functional dependencies in different records. Many functional dependencies can be implicitly represented in a database that allows the declaration of some attributes as a primary key. Most DBMS have some form of language constructs for expressing integrity constraints.

In a statistical database the objective is to maximize sharing statistical information and yet preserve privacy of individual records. This security problem cannot be solved by normal access control strategy, since the aim of the database is to allow all users full access to the data. The means to prevent compromising a statistical database is to reject queries if the number of intersection records with previous queries made by the user is very large (or very small). If random falsification is used to protect confidentiality, it is statistically insignificant, so that normal users will not suffer from erroneous statistics. The maintenance of audit trails could discourage unscrupulous snooping.

The auditing process is also relevant in nonstatistical databases to verify if the automated operations are properly implemented and executed.

Key Terms

privacy
database security
database integrity
semantic integrity
authorization
security administrator
authorizer
open system
closed system
least privilege
need to know
maximize sharing
content-independent access control
name-dependent access control
content-dependent access control
access type control
access context control
access matrix
subject
object
propagate access control
authorization grant tree
password
identification
authentication
Caesar code
one-time code
Data Encryption Standard (DES)
trapdoor functions
public key
domain integrity
implicit constraint
referential integrity
auxiliary procedure
trigger
assert
on clause
statistical database

Exercises

13.1 Consider a case of computer-related fraud you are familiar with (or consult one of the references cited in the bibliographic notes). List the security and integrity constraints that

Chapter

14

Database Design

Contents

Database design is an iterative process. A number of design methodologies have been developed.This chapter offers an informal discussion of the steps involved in designing a database.

14.1 The Organization and Its Information System

The information system of an organization consists of a number of subsystems involved in the collection, dissemination, and management of information. Some of these subsystems are manual, others are automated. A database system consisting of the data, DBMS software, hardware, and personnel is a component of such an information system.

Deciding to use a database system requires studying the organization and its needs. In the case of a small organization with few users, where the volume of data is small and there is no need for online query or update, a database system may not be necessary. In an organization with a large volume of data that changes rapidly, where there is a need for interactive queries and modifications, with a large number of users, and where decision making is distributed, the need for concurrent access to shared data is addressed by a database system.

In an organization where a large number of users and applications exist, the database system provides data independence, insulating these users and applications from changes. For the database to meets its objectives, its design must be complete and consistent. All the significant inputs should be used in the design process, including the inputs of the users. The external schema allows multiple views of the data contained in the database. Designing a database system requires gathering details about the applications and transactions that are to be supported and the classes of users that will use the system.

Figure 14.1 gives the system cycle for the design of a database system. It starts off with the definition of the problem and goes through a number of steps, culminating in the installation and operation of the system. In the following sections we examine the activities performed in each phase of this cycle.

14.2 Phase I: Definition of the Problem

The first step in the system cycle is the rough outline and scope of the project. Alternatives are examined and one of the alternatives is targeted for a feasibility study. Estimates of the costs, including initial setup and operational costs, and the risks versus the benefits are examined. The initial cost consists of acquiring the software and the hardware, converting from a manual or file-based system, and training the personnel. Time scales for the various stages of the development cycle are estimated. Approval of top management for a go-ahead is required

Once it is decided that the organization wants to pursue the database solution for its information needs, the design of the database system begins.

- **Meetings with user groups:** One or more key users from each user group is invited to provide input in determining the data and processing needs for the group. A formal interview may be supplemented by a questionnaire.
- **Analysis of the procedures and information flow:** The information gathered is analyzed for consistency and problem areas are targeted for further study.
- **Modifications to improve efficiency:** Modification in the current procedures that could improve efficiency may be discovered. Such modifications have to be discussed with the groups concerned to elicit their cooperation.
- **Preparing the initial proposal and requirement specifications:** The initial proposal is prepared and may be discussed with the user group for any omissions and corrections made to the proposed requirements.

The output of this phase is:

- Data requirements
- Properties and interrelationships of the data
- Operation requirements
- Significant events and the operations and conditions causing transitions
- Constraints

The application programs and transactions are designed at this stage of the design process. The structure of the programs, their functions, and data needs (read and write sets) are determined, and the user interface is defined.

14.4 Phase III: Preliminary Design

A preliminary design of the proposed system is derived in the next step. This design is evaluated against the initial requirements. The users are consulted and required changes are made to the design.

The cycle of the steps consisting of the definition of the problem, procedure analysis, and preliminary design is repeated until a satisfactory design is obtained.

The design of the conceptual schema is initially DBMS independent and allows a better understanding of the information requirements and their interrelationships. It describes the contents of the database without reference to its implementation. It can be understood by the nonspecialist and can be used in documenting the proposed database. A data model such as the E-R model may be used for its graphical nature, simplicity, and expressiveness.

The requirement specifications would have established the entities and the relationships among them, as well as their attributes. The primary key of the entities, the cardinality of the relationship, and constraints are to be specified in the conceptual schema design.

Structure constraints such as normal forms for relations have to be enforced by the design. Two approaches to the design of the conceptual schema may be taken: **centralized schema design** or a **view-integration approach.** In the former, the requirement specifications for each class of users are merged into a single set of specifications. The conceptual schema is designed from this single set. Any conflicts that may exist in the individual requirement specifications are to be arbitrated by the

DBA. After designing the conceptual schema, the views of the user classes are defined.

In the view-integration approach, the requirement specifications for each class of users form the basis for designing their views. These views are then integrated into the conceptual schema for the database. Conflicts such as synonyms and homonyms are easy to resolve. Conflicts that cannot be resolved by a conceptual schema to view mapping have to be arbitrated. An instance of such a conflict is where one application uses a locally generated sequential employee number and another may use the social security number, conflict in which one application views an attribute such as *Length* as meters and another interprets it as yards may be easier to resolve.

Once such conflicts are resolved the views are appropriately integrated. The integration could be stepwise, where we start by integrating two similar views. Subsequently, at each step, an additional view is merged into the integrated conceptual schema. After the conceptual schema is defined, the individual views are defined.

In the **top-down approach** to conceptual schema design we start with the major entities of interest, their attributes, and their relationships for the database application. We add other attributes and may decide to split up the entities into a number of specialized entities and add the relationships among these specialized entities.

In the **bottom-up approach,** we start with a set of attributes. We group these attributes into entities and relationships among them. We also attempt to find higher level entities to generalize these entities and locate relationships at this higher level.

The processing requirements in the form of applications and transactions are designed and their response requirements are estimated. To determine the performance requirements, the data items to be read and written out (read and write sets) for each operation in the transaction or application have to be determined. This is used to derive the number and size of input/output for each transaction and application. The performance requirements for the system will influence the distribution of files on physical devices, the physical file structure, and the need for indexes.

14.5 Phase IV: Computing System Decision

This decision may be based on the existing environment. If the database is to be implemented on an existing computer system, the choice is limited to that for the DBMS. The existing system must be able to meet the storage and processing needs of the proposed DBMS. DBMSs are usually chosen from one of the commercial systems because of the cost of developing an in-house system. Features provided by different systems are also important. Some that should be considered are report generation facilities, utilities such as menu and form-based user interface, features to support distribution of the database, communication facilities, and the like. Other considerations such as the expertise of the personnel and their preferences also come into play

The structure of the data dictates the data model of the database. If the data is mainly hierarchical, the hierararchical model and a hierarchical DBMS may be appropriate. If the data exhibits a large number of interrelationships, the network or relational model would be preferable. Deciding on a model also narrows the choice of commercial DBMSs. Other factors that can influence the choice of a DBMS are

the experience of the personnel, reputation of the vendor, and the availability of services from the vendor. Selecting the DBMS also dictates the data model.

New applications are increasingly implemented on relational DBMSs and non-relational DBMSs are retrofitted with a relational interface. The current trend for the same DBMS being able to run on different CPUs under different OSs allows some degree of independence between the choice of a DBMS and that of a computer system.

If the database is to be implemented on an existing system, it must be able to meet the processing requirements for the foreseeable features.

Factors that have to be considered in the choice of the computing system are capital costs, conversion and initial training costs, operating costs including those for personnel, and maintenance of the hardware and software.

14.6 Phase V: Final Design

The preliminary design of the database in Phase III is in database-independent form, for instance, using the E-R model. Once the DBMS is chosen, the (DBMS-independent) conceptual scheme is translated into the DBMS specific conceptual scheme and the views of the applications are derived from it as external views. The schemes are generated as programs in the DDL of the target DBMS.

The first step is to convert the conceptual and external schemes in the model of the database. We discussed the method of converting a design from the E-R model to one of the relational, network, or hierarchical models in Section 2.9. These conversions rules are summarized in Figure 14.2.

14.6.1 Designing the Conceptual Database—Relational DBMS

It is apparent from Figure 14.2 that converting the preliminary design in the E-R model to a relational model is a trivial task. An entity type is represented as a relation. A weak entity type is represented as a relation that includes the key of the identifying strong entity. A relationship is also represented as a relation and includes the primary keys of the entities involved in the relationship. In the case of a 1:N relationship, if it does not involve any attributes and if the entity on the "N side" does not participate in any other relationships, the 1:N relationship can be represented by appending the primary key of the "1 side" to the relation for the "N side." It is also possible to merge these two relations into one if performance requirements are not compromised.

An *IS_A* relationship representing a generalization-specialization hierarchy (superclass/subclass relationship) in the E-R diagram may be represented as a set of relations. Here a relation is created for the superclass entity and its key is used as a foreign key in each of the relations corresponding to the subclass entities. Another option is to have the subclass entities inherit the attributes of the superclass entity. (These options were illustrated in Figures 2.28 and 2.30, respectively.) In a third option, a single relation is created that includes the attributes of the entities at all levels of the generalization-specialization hierarchy. In this case null values are used