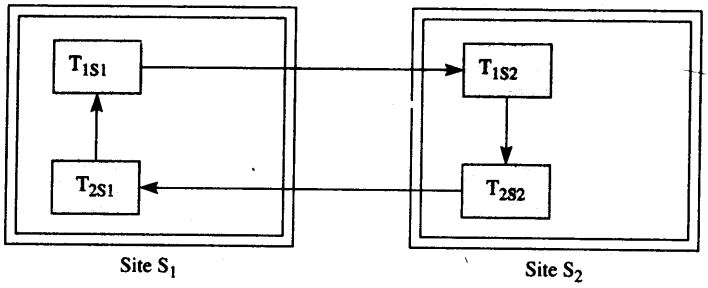
Figure N A global wait-for graph.



locking protocol and the schedule shows that the transactions will be deadlocked. The global wait-for graph for the situation at step s_5 of the schedule is shown in Figure N. ■

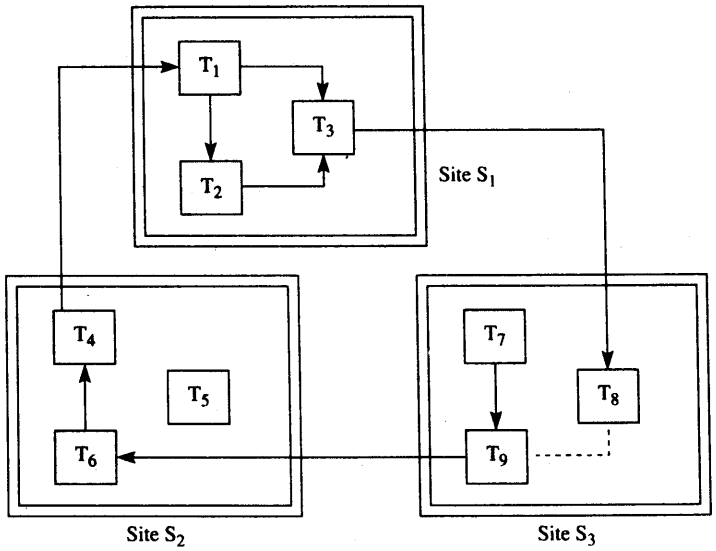
One simple method of recovering from a potential deadlock situation is to allow a transaction to wait for a finite amount of time for an incompatibly locked data item. If at the end of that time the resource is still locked, the transaction is aborted. The period of time should not be too short or too long. An unduly short period would likely cause the transaction to be aborted, since the resource may not be released. An unnecessarily long period would mean that these transactions would hold the resources already acquired, causing further transactions to deadlock. With this scheme, only transactions that are blocked are aborted.

The deadlock detection scheme allows deadlock to occur, but makes provision to detect the existence of a deadlock by the presence of a chain of transactions, each waiting for data items locked by the next transaction in the chain. The detection of deadlock in a distributed system requires the generation of not only a local wait-for graph (LWFG) for each site, but also a global wait-for graph (GWFG) for the entire system. Note that here we are assuming that a transaction can request one or more data items at a time and become blocked if it has at least one outstanding request for a data item. Under this assumption, a cycle in the global wait-for graph indicates a deadlock situation. Figure N shows the GWFG for the execution schedule of Figure M.

We see from Figure N that even though there are no cycles in the LWFG at each of two sites, there is a cycle in the GWFG and this indicates the existence of a deadlock. The disadvantage of the GWFG is the overhead required in generating such graphs. Furthermore, a deadlock detection site has to be chosen where the GWFG is created. This site becomes the location for detecting deadlocks and selecting the transactions that have to be aborted to recover from deadlock. One of the problems with such an approach is that if the messages indicating which transactions are waiting for which resources and the release of the resources by transactions are received out of order, then the deadlock detection site may conclude that there is a deadlock. However in reality no such deadlock exists. The erroneous deadlock that was detected is called a **phantom deadlock.** Example 15.16 shows how a phantom deadlock could result.

Example 15.16 Consider the GWFG of Figure O. Suppose the graph is maintained at site S. Suppose there is a request from T_8 for a data item locked by T_9 at about

Figure 0 A global wait-for graph.



the same time as T_4 releases the data item it locked. The release of the data item allows T_6 to proceed and causes a removal of the edge (T_6,T_4). If the fact that the edge (T_6,T_4) is removed reaches site S after it learns of the addition of the edge (T_8,T_9), then a phantom deadlock would be detected. The cycles $(T_1,T_3,T_8,T_9,T_6,T_4,T_1)$ and $(T_1,T_2,T_3,T_8,T_9,T_6,T_4,T_1)$ do not, in fact, exist. ■

Instead of using a central site for deadlock detection, it is possible to use a distributed deadlock detection scheme. In one such approach, the LWFGs are broadcast to all sites. Each such site generates the portion of the GWFG that is of concern to it. If one of the site detects a deadlock, it tries to resolve it by aborting one or more of its transactions. The disadvantage of this approach is that the deadlock may not be detected for some time, and since the broadcast of the LWFGs is asynchronous, a phantom deadlock could be detected and lead to unnecessary transaction aborts.

Below we give another scheme, known as a probe computation algorithm, for distributed deadlock detection. It is from a class of algorithms called the **edge-chasing algorithms.** For other deadlock detection algorithms, refer to the bibliographic notes.

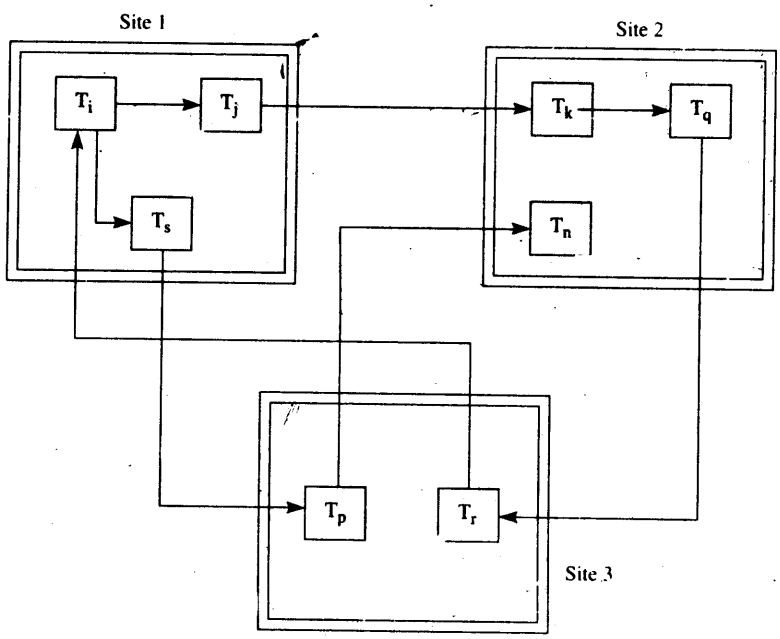
15.9.1 Deadlock Detection by Probe Computation

In the edge-chasing algorithms, the cycle in the GWFG is detected not by actually creating the graph but by sending messages along the edges of the graph. Such messages, called **probes,** are different from the other messages discussed above, and

also distinct from resource requests and grant messages. As before, instead of referring to transactions, let us use the process concept, which encompasses the interaction between a transaction at a site with the TM at the site. Figure 15.9 shows an example of a number of such processes in a GWFG in a deadlock situation. Let us examine how probe computation is used to detect the deadlock.

An edge from a process in one site to a process in a distinct site is called an **intercontroller edge.** An **outgoing edge** for a process is an intercontroller edge that can be reached from the process by following edges in the local part of the GWFG. The probe is initiated by a blocked process and it is referred to as the initiator of the probe. A probe is made up of a three-tuple (i, j, k) and indicates that it is a probe for process T_i and the probe has been sent along the outgoing edge (T_j, T_k). Here process T_j is blocked by process T_k and T_i is blocked, directly or via a chain of intermediate processes, by T_j. If the initiator of the probe receives a matching probe, we can conclude that the blocked process is in a cycle in the GWFG. Thus, if process T_i, the initiator of a probe, receives a probe (i, x, i), it is in a cycle. An active process simply discards the probes. A blocked process propagates the probe along all its outgoing edges. This blocked process will send a probe (i, j, k) to the process at node k, along outgoing edge (T_j, T_k), under the following conditions: (a) the process T_j is blocked, (b) T_j is waiting for the process T_k, (c) T_i is blocked by T_j. Note that a site that has several blocked processes may initiate several probes. Similarly, several probes may be initiated in sequence by a blocked process if it has several outstanding requests. Each such probe is distinctly identifiable by the initiator.

Figure 15.9 Detection of deadlock using probe computation.



A blocked process, T_k, accepts only meaningful probes; others are discarded on receipt. A probe (i,j,k) is said to be meaningful under the following conditions: (a) the process T_k is blocked, and (b) it was unaware that T_i was dependent on it (that T_k is in a chain with T_i). Condition (b) ensures that nonmeaningful probes are suppressed and, consequently, only one probe per cycle is propagated. Receipt of a meaningful probe leads to the deduction that T_i is dependent on T_k and further probes (i,j,k) will be discarded (they will not longer be meaningful). On receipt of a meaningful probe, T_k sends, probes (i,p,q) on all its outgoing edges (T_p, T_q).

If a blocked process, T_i, receives a probe (i,j,i), we can deduce that it is deadlocked. The **probe computation** algorithms are given below and Example 15.17 illustrates its use in the detection of a distributed deadlock.

Algorithm 15.1 **A Blocked Process Initiates Probe Computation**

if process T_i is blocked and the deadlock is not local,
  then for each distinct outgoing edge T_j, T_k
    transmit a probe (i,j,k)

Algorithm 15.2 **A Blocked Process Response to Probes Received**

for each probe (x,y,z) received by T_i
  if probe is meaningful
    then if x = z (both x and z also being equal to i for some blocked process T_i)
      then T_i is in a deadlock: initiate deadlock resolution
      else for each distinct outgoing edge T_p, T_q
        transmit a probe (x,p,q)

Example 15.17

Consider the situation depicted in the GWFG of Figure 15.9. We have processes T_i, T_j, T_k, T_q, T_r, T_i in a deadlock situation. The probe is initiated, say, by T_i. It sends a probe (i,j,k) along its outgoing edge (j,k) to process T_k at site 2. When T_k receives this probe it finds it meaningful since it was unaware that T_i was blocked by it. It knows that T_j is blocked by it since it has not released the data item requested by T_j. T_k, in turn, sends a probe (i,q,r) along its outgoing edge (T_q,T_r) to T_r at site 3. T_r finds this probe

meaningful and, in turn, learns that T_i is dependent on it. T_r sends the probe (i,r,i) along its outgoing edge (r, i) to T_i. On receipt of this probe T_i learns that it is in a closed cycle in the GWFG. Note that a probe (i,s,p) along the outgoing edge (T_s,T_p) will be eventually ignored by process T_n and will not reach T_i. ■

15.9.2 Deadlock Prevention

The deadlock prevention method can be used in a distributed system. For instance, the timestamp method could be applied to prevent deadlock from occurring by aborting the transactions that could potentially cause deadlock. The wait-die scheme and the wound-wait scheme could be used to abort appropriate transactions as in the centralized system. The aborted transactions are reinitiated with the original timestamp to allow them to eventually run to completion. The timestamp method does not require that any messages be transmitted over the network; however being a deadlock prevention scheme it causes unnecessary transaction aborts.

15.10 Security and Protection

Security and protection problems are similar to those in the centralized database with remote access. However, the problem is exacerbated by the fact that there is increased communication, including site-to-site transfer of large amounts of data. This calls for appropriate identification and authentication of the user and the site. To prevent eavesdropping on the communication lines by intruders, these lines must be secure and the message should be encrypted.

The fact that data is replicated in the database means that a user can access any one of these replicated copies. Security dictates that the authorization rules for access and update of certain parts of this data be verified before user action is allowed. If the authorization rules are centralized, the authorization validation will generate traffic and the central site would become the bottleneck. Another approach is to replicate the authorization rules. Full replication allows local validation of user action at the time of compilation or execution of the user query. However, full replication adds unnecessary update overheads. Still another approach involves replicating, at a given site, only those authorization rules that pertain to the data items at the site. The maintenance problem is improved but validation of a user's action for a remote site can only be done at the remote site during an advance compilation or execution stage of the user's query. Considerable computing efforts are wasted, since a query is aborted on discovery that the query lacks authorization for particular data items.

15.11 Homogeneous and Heterogeneous Systems

In general, a distributed database system may be either **homogeneous** (i.e., all local database systems have the same underlying data model) or **heterogeneous** (i.e., local

is therefore possible for local objects to have their catalog entries at a remote site. Consequently data definition operations may be nonlocal.

SDD–1 uses a centralized control scheme and a given transaction is supervised by a single transaction module. The transaction execution is in three phases, read, execute, and write. All reads for a transaction are performed at the beginning of the transaction; all writes are done at the end.

The read-set, the set of fragments to be read by the transaction, are determined by the TM. SDD–1 provides fragmentation transparencies and the set of fragments to be accessed for the read-set of the transaction are determined by it. This operation is called **materialization.** The TM then coordinates the data modules at various sites to transfer the required data into the workspace used by the transaction.

SDD–1 uses the **conservative timestamp** method for concurrency control. We briefly describe the conservative timestamp scheme here. The basic timestamp method, discussed in Chapter 12, suffers from costly restarts.

A pessimistic approach is taken by the conservative timestamp method, which causes possible delays until no conflict with a transaction can possibly happen. This is done by buffering younger transactions until all possible older transactions have been executed. At some point the system must decide that no older transaction is likely to be received. A simple implementation of the idea requires that a site send all requests to another site in timestamp order and that the network deliver messages in the order that they were sent. In this manner, if site j receives a message from site i with a timestamp t_{ik}, site j knows that it will not receive messages from site i with timestamps less than t_{ik}.

At site j, instead of keeping the timestamps of the last read or write operation on data item X, we now keep the oldest timestamp for the buffered read and write operations for data item X, say X_{RMIN} and X_{WMIN}, respectively. Site j, having received messages from all other sites, can proceed to execute an operation. The method is implemented as follows:

while not at least one request from each site wait.
READ
    if $X_{RMIN} < X_{WMIN}$
        then the read is older than any of the write requests on X, i.e., all older transactions that could update X have been executed and thus the read operation can be executed and X_{RMIN} updated.
        else there is at least one older update request and the read is buffered.
WRITE
    if $X_{WMIN} < X_{RMIN}$
        then all read requests for the current data value have been executed and the write operation can be executed; the value of X_{WMIN} is updated.
        else there is at least one older transaction that has still to read the current value and thus the write remains buffered.

This simple method suffers from the fact that at least one request must be received from each site. So that the system will not remain blocked, each active site

that lacks an action request sends a null request message. In this manner, the buffer from each site would at least have a request, albeit null.

Major disadvantages of the conservative timestamp method are the long waiting periods and consequent low concurrency. The improvements suggested to overcome these shortcomings include the use of **transaction classes** and **conflict analysis graphs.**

All transactions are categorized into classes. Those transactions that are likely to conflict belong to the same class. A transaction's read requests can be termed its read-set and the write requests, its write-set. Two transactions conflict if the intersection of their write-sets or the read-set of one and the write-set of the other is non-empty. For instance, let the read and write sets of transaction T_1 be RS_1 and WS_1, respectively, and those for T_2 be RS_2 and WS_2. Then transactions T_1 and T_2 conflict if

$$RS_1 \cap WS_2 \neq \emptyset, \text{ or}$$
$$WS_1 \cap RS_2 \neq \emptyset, \text{ or}$$
$$WS_1 \cap WS_2 \neq \emptyset$$

Transaction requests waiting in buffers need only be compared with requests from conflicting transactions. Two requests from nonconflicting transactions can proceed concurrently, thus improving concurrency.

Whether or not transactions conflict can be decided using conflict graph analysis techniques in which arcs can be labeled to define the type of conflict (read-write or write-write).

The execute phase is performed by a compile-and-go approach. The access plan generated is executed and the supervision is by the TM at the site of the query. Program compilation uses semijoins extensively in optimization. The write phase begins by the distribution of the updated fragments to all data modules containing a replicate of an updated fragment. The updated fragments are made permanent by write command issued by the coordinating TM.

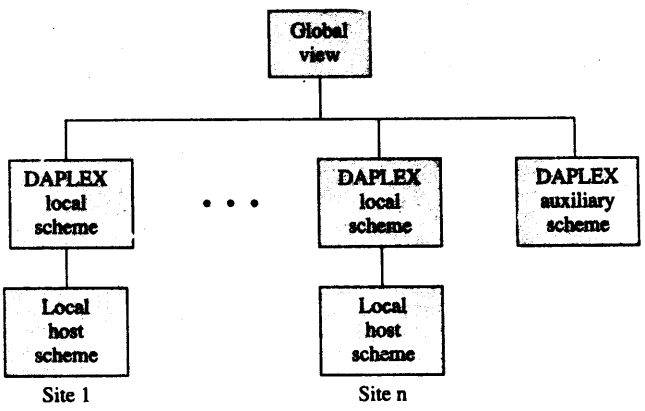
Transaction atomicity is provided in SDD–1 by a four-phase commit protocol, which includes selection of backup coordinators to supervise the commit protocol in case of coordinator failure.

R* (see Figure 15.12) is an experimental adaptation of the System R relational DBMS to the distributed environment. The architecture of R* is based on System R architecture. It is claimed that major modifications were made to the relation data storage (RSS*) and transaction manager (TM*) systems. A distributed communications (DC*) component was added.

R* runs under IBM's Customer Information Control System (CICS). CICS is responsible for handling online users and could entail running application programs or provide support for interactive queries. CICS is also responsible for intersite message communications and interfaces with another CICS at a remote site.

All requests are made at a single site, which becomes the **master site.** In common with System R, queries are compiled rather than interpreted. A distributed compilation scheme is used wherein the master site coordinates the global aspects of query compilation. The local decisions, including local data structure selections, are

Figure 15.14 MULTIBASE scheme architecture (adapted from [Land 82]).



In a heterogeneous system, the level of performance of the various host DBMSs may differ considerably. For instance, the same operation on the same data may be done at varying costs under different DBMSs. This factor would be a consideration in allocating subtransactions, just as the nearness criteria is used when considering communication costs. In some cases, a particular host DBMS may not even be able to perform a specific operation. The above issues are addressed below in the section on MULTIBASE. If the DDBMS provides a single interface to external users, as is the common practice, then the network DML should be easy to learn and sufficiently powerful to satisfy all needs.

A few heterogeneous DDBMS prototypes have been built. Here we consider only one of them, a derivative of SDD–1.

Multibase

The MULTIBASE DDBMS,[2] developed by Computer Corporation of America, provides an integrated interface for retrieving data from preexisting, heterogeneous, distributed databases. Its aim is to present an integrated uniform interface. This is achieved by defining an integrated global schema and by utilizing a single global query language. Besides being read-only, MULTIBASE does not implement controls to ensure that when reading data from one site, other required data at another site is not being updated—because most systems do not make concurrency control services available to an external process. The global query language DAPLEX is based on the functional data model.[3] This model consists of entity sets and functions between them, and models object types of concern and their characteristics. The schema architecture is shown in Figure 15.14.

Resolving data and naming inconsistencies and any other incompatibilities in the preexisting databases are functions of the MULTIBASE system. For this reason an

[2]The discussion here is based on (Land 82).
[3]See (Ship 81) for details on DAPLEX and the functional data model.

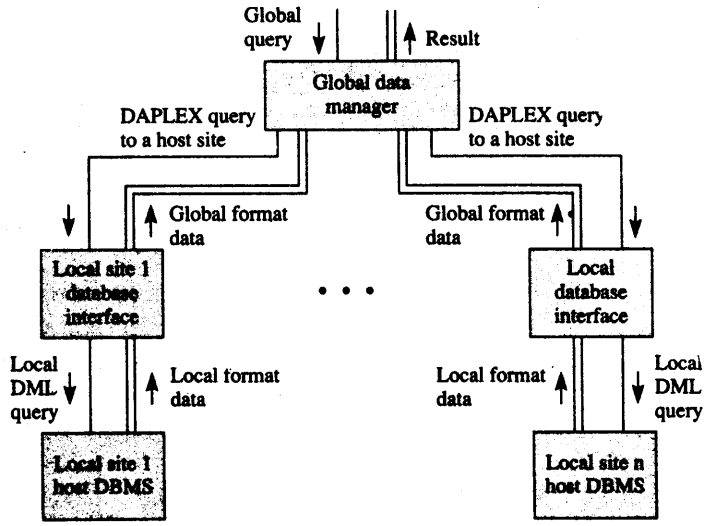
integration schema called the auxiliary schema is specified. The auxiliary schema describes a DAPLEX auxiliary database that is maintained by an internal DBMS, which is part of MULTIBASE. It contains data unavailable in any of the host DBMSs, or data needed to solve incompatibilities. Examples of such data are the following: statistics to determine which data values should be used in case of conflict; conversion tables for performing data transformations that can't be done via simple arithmetic manipulations. Furthermore, if two sites have, say, EMPLOYEE data but only one site has an EMPLOYEE.*Phone_No*, the missing phone number data can be added to the auxiliary database. This is in addition to the global and local schema (both specified using DAPLEX) and a local host schema. The local host schema is the schema description in the local DBMS language.

Besides the language and schema definition systems, MULTIBASE also provides a query processing system. The query processing system incorporates the local data interface and local DBMS. At the global level are the query translator and query processor subsystems. The query translator transforms the global query into subqueries over the local and auxiliary schema. The global query processor chooses appropriate query optimization criteria and coordinates the local query executions. The optimization plan includes data movement between sites and the integration of the results from the sites.

Local queries are sent to the local sites and are subjected to local query optimization. These locally optimized queries are then translated into queries over the local host schema of the host DBMS.

Each of these tasks is performed at different levels. Figure 15.15 displays the MULTIBASE architecture, which shows the two major components, the **global data manager (GDM)** and **local database interface (LDI).** The user submits queries to the global data manager, which is responsible for global query translation and optimization. It receives results from the local sites and performs any processing necessary to output the result. At each local site is a local database interface module that

Figure 15.15 MULTIBASE architecture (adapted from [Land 82]).



appropriate modifications. Detection of deadlock requires the generation of global wait-for graphs, either directly or indirectly. The deadlock prevention method based on schemes for a centralized system may be used. The problems of security and protection are similar to those of a centralized system.

Key Terms

distributed database
global transaction
subtransaction
local transaction
distributed database management system (DDBMS)
wide area network
long haul network
local area network (LAN)
star topology
mesh connection
bus network
ring topology
synchronous time-division multiplexing (SDM)
time-division multiplexing (TDM)
frequency division multiplexing (FDM)
polling
token
reservation
ALOHA
carrier sense multiple access with collision detection (CSMA/CD)
Ethernet
star ring
bridge
gateway
network partitioning
replication
location transparency
fragmentation
disjoint fragmentation
fragmentation transparency
vertical fragmentation
horizontal fragmentation
mixed fragmentation
disjoint vertical fragmentation
disjoint horizontal fragmentation
nondisjoint vertical fragmentation
nondisjoint horizontal fragmentation
replication transparency
update transparency
system catalog
birth_site
global catalog
locator catalog
caching the remote catalog
global naming
print name
systemwide name (SWN)
semijoin
reduction of relations
query graph
cyclic query
coordinator site
participating site
transaction manager
invariant
local wait-for graph (LWFG)
global wait-for graph (GWFG)
majority locking
primary site locking
distributed two-phase locking
distributed commit
blocking
two-phase commit
voting phase
decision phase
phantom deadlock
edge-chasing algorithms
probe
intercontroller edge
outgoing edge
probe computation
homogeneous system
heterogeneous system
local scheme
global scheme
materialization
conservative timestamp
transaction class
conflict analysis graph
master site
apprentice site
global data manager (GDM)
local database interface (LDI)
transformer
optimizer
decomposer
filter
monitor

Exercises

15.1 Explain why the query processing techniques discussed in this chapter would need to be modified for a distributed system running on a local area network. In your opinion, which of the three costs, communication, I/O, or CPU, are likely to dominate in a local area network environment? Justify your answer.

15.2 What are the advantages of horizontal fragmentation? How is query evaluation complicated or simplified by horizontal fragmentation? Design an algorithm to perform the join of two relations, R and S, both of which are horizontally fragmented. Account for the network to be either wide area or local area. Create some arbitrary data for the relations and their fragments. Distribute the fragments over a number of sites. Test your algorithm.

15.3 For exercise 2, modify your algorithm to use the semijoin technique.

15.4 Under what conditions is $R \ltimes S = S \ltimes R$?

15.5 How can the optimistic method presented in Chapter 12 be applied to concurrency control in a DDBMS? Discuss the relative advantages and disadvantages of the conservative timestamp and optimistic methods.

15.6 The validation phase of the optimistic method of a transaction may be checked against already committed transactions—the "committed validation technique," or the currently active (but not committed) transactions—the "active validation technique." Discuss the relative merits of these validation techniques for the optimistic concurrency control scheme for a DDBMS.

15.7 Using the library example discussed in Chapters 8 and 9, create a suitable distributed database. Indicate how the queries in those chapters would be handled.

15.8 Suppose a single copy of data items A and B is stored at sites S_1 and S_2, respectively. Consider the schedule for transactions T_1 and T_2 given in Figure P. Why is the schedule serializable, even though two-phase locking is not used?

Figure P Schedule for Exercise 15.8.

	site S_1		site S_2	
Time	Transaction T_{1S1}	Transaction T_{2S1}	Transaction T_{1S2}	Transaction T_{2S2}
t_1				
t_2	**Lockx**(A)			**Lockx**(B)
t_3	**Read**(A)			**Read**(B)
t_4	A := A − 100			B := B − 200
t_5	**Write**(A)			**Write**(B)
t_6	**Unlock**(A)			**Unlock**(B)
t_7		**Lockx**(A)	**Lockx**(B)	
t_8		**Read**(A)	**Read**(B)	
t_9		A := A + 200	B := B + 100	
t_{10}		**Write**(A)	**Write**(B)	
t_{11}		**Unlock**(A)	**Unlock**(B)	

15.9 Consider a token approach to locking. Any number of read tokens can exist for a data item, but only one write token can exist, and that only if no read tokens are present. A transaction manager (TM) at a site can grant a read or write lock to a transaction at that site if the TM has a read or write token for the data item. Indicate the sequence of messages required between sites to allow transaction T running at site S to obtain a write lock on data item A.

15.10 Consider the following scheme to detect deadlock in a distributed database system. Each site maintains an LWFG with the addition of a node called T_{ex} (see Figure Q). T_{ex} is to depict

(Eswa 76) K. P. Eswaran, J. N. Gray, R. A. Lorie, & I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," *Communications of the ACM* 19(11). November 1976, pp. 624–633.

(Gray 75) J. N. Gray, R. A. Lorie, G. R. Putzula, & I. L. Traiger, "Granularity of Locks and Degrees of Consistency in a Shared Database," IBM Research Report RJ1654, 1975.

(Gray 81) J. N. Gray, "The Transaction Concept: Virtues and Limitations," Proceedings of the Seventh International Conference on Very Large Databases, 1981, pp. 144–154.

(Heve 79) A. R. Hevener & S. B. Yao, "Query Processing in a Distributed Database System," *IEEE Transactions on Software Engineering* SE–5(3), May 1979, pp. 177–187.

(Knap 87) E. Knapp, "Deadlock Detection in Distributed Databases," *Computing Surveys* 19(4), December 1987, pp. 303–328.

(Land 82) T. Landers & R. L. Rosenberg, "An Overview of Multibase," in H. J. Schneider, ed. *Distributed Data Bases*. New York, North Holland, 1982, pp. 153–188.

(Nava 84) S. Navathe, S. Ceri, G. Widerhold, & J. Dou, "Vertical Partitioning Algorithms for Database Design," *ACM Transactions on Database Systems* 9(4), December 1984, pp. 680–710.

(Papa 85) C. H. Papadimitriou & M. Yannakakis, "The Complexity of Reliable Concurrency Control," Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems, 1985, pp. 230–233.

(Papa 86) C. H. Papadimitriou, *The Theory of Concurrency Control*. Rockville, MD: Computer Science Press, 1986.

(Roth 80) J. B. Rothnie, P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, & E. Wong, "Introduction to a System for Distributed Databases (SDD–1)," *ACM Transactions on Database Systems* 5, January 1980, pp. 1–17.

(Ship 81) D. W. Shipman, "The Functional Data Model and the Data Language DAPLEX," *ACM Transactions on Database Systems* 6(1), March 1981, pp. 140–173.

(Smit 81) J. M. Smith, P. Bernstein, U. Dayal, N. Goodman, T. Landers, K. W. T. Lin, & E. Wong, "Multibase-Integrating Heterogeneous Distributed Database Systems," NCC Conf. Proc. 50, May 1981, pp. 487–499.

(Stal 87) W. Stalling, *Local Networks, An Introduction*, 2nd ed. New York: Macmillan, 1987.

(Ston 77) M. Stonebraker & E. Neuhold, "A Distributed Database Version of INGRES," 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, University of California, Berkeley, 1977, pp. 19–36.

(Will 81) R. Williams, D. Daniels, L. Haas, G. Lapis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, P. Wilms, & R. Yost, "R*: An Overview of the Architecture," IBM Technical Report, RJ 3325, San Jose, CA, 1981.

(Wong 77) E. Wong, "Retrieving Dispersed Data from SDD–1: A System for Distributed Databases." Proc. of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, CA, 1977, pp. 217–235.

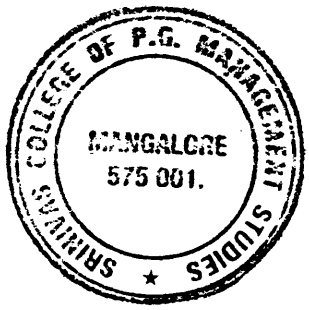
(Yu 84) C. T. Yu & C. C. Chang, "Distributed Query Processing," *ACM Computing Surveys* 16(4), December 1984, pp. 399–433.

Contents

Chapter

16

Current Topics in Database Research



In a knowledge base system, the emphasis is placed on a robust knowledge representation scheme and extensive reasoning capability. **Robust** signifies that the scheme is rich in expressive power and at the same time it is efficient. In a DBMS, emphasis is on efficient access and management of the data that model a portion of the real world. A knowledge base system is concerned with the meaning of information, whereas a DBMS is interested in the information contained in the data. However, these distinctions are not absolute.

For our purposes, we can adopt the following informal definition of a KBMS. The important point in this definition is that we are concerned with what the system does rather than how it is done.

Definition: A knowledge base management system is a computer system that manages the knowledge in a given domain or field of interest and exhibits reasoning power to the level of a human expert in this domain.

A KBMS, in addition, provides the user with an integrated language, which serves the purpose of the traditional DML of the existing DBMS and has the power of a high-level application language. A database can be viewed as a very basic knowledge base system insofar as it manages facts. It has been recognized that there should be an integration of the DBMS technology with the reasoning aspect in the development of shared knowledge bases. Database technology has already addressed the problems of improving system performance, concurrent access, distribution, and friendly interface; these features are equally pertinent in a KBMS. There will be a continuing need for current DBMSs and their functionalities coexisting with an integrated KBMS. However, the reasoning power of a KBMS can improve the ease of retrieval of pertinent information from a DBMS.

16.3 Knowledge and Its Representation

To solve a problem (i.e., carry out an intelligent activity) we need three components:[3]

- A model or a symbolic representation of the concepts of the domain of interest.
- A set of basic operations on this symbolic representation to generate one or more solutions to the problem.
- An evaluation method to select a solution from the set of possible candidates.

The representation scheme must be able to register the significant characteristics of the problem domain. These features of the problem domain must be easily accessible for appropriate manipulations.

[3]A. Newell & H. A. Simon, "Computer Science as Empirical Inquiry: Symbols and Search," *ACM 10th Turing Lecture—1975, CACM* 19(3), March 1976, pp. 113–126.

A natural language is an example of a symbolic representation scheme. Using this scheme, knowledge has been represented in folklore and more recently using the written word and recorded speech and image. These forms of representing knowledge have been developed over thousands of years. Natural language has very high expressive power. However, this form of representation, though suitable for humans, is either inappropriate or requires an enormous amount of resources for use in a computer-based system.

A knowledge base system contains knowledge about a particular domain. In addition, it contains a certain amount of general knowledge. The latter includes the pertinent world knowledge applicable to the domain and some degree of so-called commonsense knowledge. For instance, a knowledge base system containing information about diseases and diagnoses must have knowledge of the different units of measurements of mass, length, temperature, concepts of nearness, normal, higher, lower, faster, slower, and so on.

Just as beauty is in the eye of the beholder, meaning is not contained in the message, but is constructed around it by the recipient. For example, if we are presented with the statement "Jumbo is an elephant," we conjure up a picture of an elephant; we know that it is large, with a trunk and tusks and huge flapping ears. In order to make this addition to the simple statement, we recalled this common knowledge that we acquired during our life. If we are then presented with the statement "Jumbo lives in a teacup," we will think either (a) the statements are from a fairy tale or (b) the statements are inconsistent with what we know about elephants and cups in the real world. We know from experience that elephants are large animals, a normal teacup is too small to hold an elephant, and normally we don't put elephants in teacups!

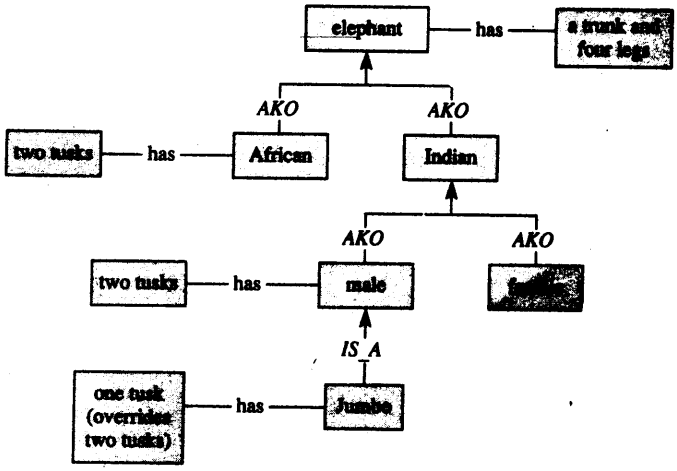
One of the requirements of any knowledge representation scheme is that it must allow the associated knowledge about a concept or statement to be easily retrieved and employed to enable the knowledge base system to understand and reason. The concept of using association in retrieving information is a very old one; it can be traced back to the time of Aristotle. The use of association in database applications in the form of associative or intelligent memories has also been investigated. However, the use of associative memory[4] to model human memories for intelligent computer systems is more recent. The efficient access to associated knowledge in a particular situation need not be in a form similar to human memory; nevertheless, the result should be useful so that related concepts, associated both explicitly and implicitly, can be employed in inferences.

We not only know something, we know that we know it and have developed a certain degree of confidence in using the knowledge correctly (expertise). Our ability to read a map, our sense of orientation, and knowledge of these abilities give us the confidence to drive to an unknown city and find an address. Similarly, the knowledge base system must have knowledge about the knowledge representation scheme being used and how it can be manipulated in the reasoning process. Such knowledge, called **metaknowledge**, can be compared to the metadata used in a database system.

The knowledge base system must be able to deal with incomplete knowledge,

[4]An associative memory system has logic associated with each word or each bit of every word. This logic is used to simultaneously examine the contents of the entire memory and matching words are flagged.

Figure 16.2 Inheriting and overriding properties.

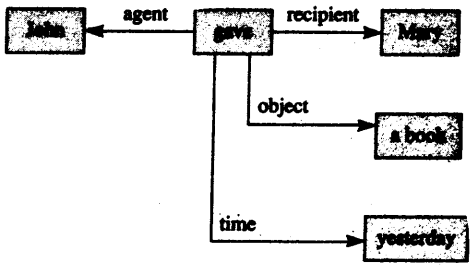


The *AKO* link can also be used to provide specialization or a classification or taxonomy in the reverse direction. In Figure 16.1 the *AKO* link provides a relationship between two generic nodes and the *IS_A* link provides it between a generic and an individual node. Thus, the generic nodes, human and mammal, are connected by an *AKO* arc to show the relationship that the human is a type of mammal. The relationship between a node representing an individual, Jumbo, and the generic node, elephant, is provided by an *IS_A* arc. In Figure 16.2, the node Jumbo inherits the properties of the elephant, Indian, and male nodes.

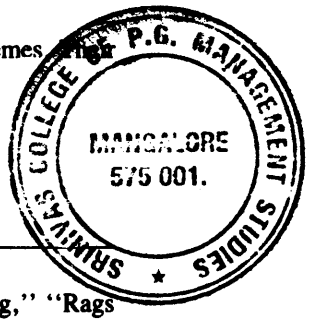
The semantics of an action, for instance, "John gave Mary a book yesterday," can be represented as shown in Figure 16.3.

In addition to assigning the meaning to the arcs in the network, the use of a semantic network for knowledge representation requires that procedures using the semantic network correctly interpret the meaning of these arcs. The assigning of meaning to the arcs is ad hoc and a wide variation of network-based schemes have been proposed, along with procedures to interpret them. In spite of a lack of stan-

Figure 16.3 Assigning a meaning to an arc.



dardization, semantic networks are popular knowledge representation schemes since their structure resembles the association perceived to be used by humans.



16.3.2 First-Order Logic (Predicate Logic)

A **proposition** is a declarative (or assertive) sentence, e.g., "It is snowing," "Rags is a dog." Declarative sentences are distinguished from interrogative and imperative sentences. An interrogative sentence asks a question; an imperative sentence is a directive or command.

Propositional logic is concerned with establishing the validity of a proposition in light of a given set of propositions. It establishes whether the proposition is true or false, relative to the given set. (The propositions in the set could be either true or false. It is, however, not in the realm of logic to establish the truthfulness of each statement in the given set of propositions.) Simple propositions can be combined using the sentential connectives and, or, not, implies, equivalent, and so on. An example of a combined proposition, which is always true, is the following: "If elephants are mammals, and if Jumbo is an elephant, then Jumbo is a mammal."

Propositional calculus is, in effect, computing with propositions. Given a set of propositions or axioms known to be true (or false), propositional calculus uses rules of inference to determine whether a given proposition is true or false. Let us use X, Y, Z, etc. to denote propositions; for instance, X may be the proposition "Jumbo is an elephant." The first rule of inference, called **modus ponens,** allows us to infer that proposition Y, "Jumbo is a mammal," is true under the condition that proposition X is true, and hence X logically implies Y (written as $X \rightarrow Y$). Thus, given that "Jumbo is an elephant" (X) is true, and "Jumbo is an elephant" implies "Jumbo is a mammal," which is also true, then "Jumbo is a mammal" is true. That is, if X and $X \rightarrow Y$ are both true, Y is also true. In the above example Y is the proposition "Jumbo is a mammal".

The second rule of inference is the **chain rule,** which allows us to infer a new implication from known implications; thus, if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Mathematical logic, just like any other formal science, uses a language to express statements or formulas. The semantics of these statements are well defined. Mathematical logic also uses a theory of proofs so that statements can be proved to be correct or false. One method of proving the truthfulness or falsity of a proposition is called **reductio ad absurdum.** In this method the known propositions or axioms are appended with the negation of the proposition to be proved. If the resulting set is inconsistent, the proposition cannot be false. A major problem with this approach is that as the number of propositions increases, the number of combinations to be investigated increases in an exponential manner and the computation time becomes astronomical.

Propositions that specify a property consist of an expression that names an individual object and an expression, called the **predicate,** that stands for the property that the individual object possesses. We use the lowercase symbols from the end of the alphabet to denote variables, those from the beginning of the alphabet to denote constants, and uppercase letters to denote predicates. Thus P(x), where x is the argument, is a one-place or monadic predicate. DBMS(x) and COMPANY(y) are ex-

In a Horn clause there is only one conclusion. PROLOG (PROgramming in LOGic) is a programming language based on the Horn clause. The human reasoning process is generally considered to be similar to the scheme used in logic. Logic results in a precise and flexible knowledge representation scheme that is easy to formulate and understand. The disadvantages of using logic are the lack of indexing or associative capability, the handling of dynamic and incomplete knowledge, and the intractability of computations involved in logic-based deductive inferences. In a logic-based knowledge representation, the processing is separated from the knowledge representation. The processing part, which determines the utility of the system, is usually implemented by theorem-proving techniques; however, this approach may not be useful for all applications. Another drawback of logic-based representation schemes is that heuristic or rule-of-thumb type knowledge may not be expressed in logic. In addition, the following assumptions have to be made in this processing (Gall 84), (Reit 84):

- **Closed world assumption (CWA),** which states that facts not known to be true are false.
- **Unique name assumption (UNA),** which states that objects are uniquely identified.
- **Domain closure assumption (DCA),** which states that no other objects or instances of objects other than the known ones exist.

16.3.3 Frames

The frame is another knowledge representation scheme used to represent the knowledge from a limited domain of stereotyped concepts or events. The concept of frames evolved from observations gleaned by psychologists as to the method humans use to interpret new situations. When confronted with an unknown object from a category of objects already experienced, we expect certain similarities and accept certain differences. We know how to handle these differences. We know what to expect and what to do if these expectations don't materialize. Thus, when we drive to a new city, we expect to see parks, buildings, streets, street signs. We know the usual locations of street signs and the correspondence between a street and a sign when a number of signs are posted on the same signposts. We also know what to do if a sign is missing at an intersection.

A **frame** (see Figure 16.5), is a data structure representing the collection of the expected and/or predicted description of a stereotype object, action, or event. Each important feature of the object is held in a slot. An optional procedure can be attached to a slot to introduce procedural information or specify consistency constraints. The frame also contains the object's relationship to other objects, these being represented by frames as well. The latter feature gives a frame a semantic network–like property. The description of the object includes a number of important features of the object and the relationships between other descriptors.

In addition to the predicted description of the various features of the object being represented, the frame may contain information such as the level of confidence assigned to the descriptor, the default values, alternate values (or their range) for descriptors, and variations in the descriptors that can be associated with the frame. The descriptors or slots can allow the inheritance of properties from a related frame. In

Figure 16.5 Frames

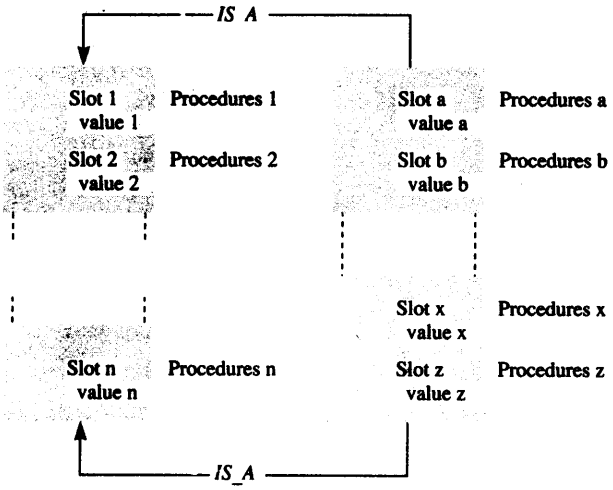


Figure 16.6, the frame *Bungalow* is a specialization of the frame *Building* and inherits from *Building* descriptors such as walls, doors, windows, roof. The slot or descriptor itself can be a frame. Thus, the descriptor *window* can have descriptors of its own, for example, *size* and *type*.

In addition, descriptors could have appropriate reasoning or inferencing procedures attached. These procedures are triggered or executed whenever the descriptors are filled in, modified, or matched to glean precompiled knowledge.

Frames have been used extensively to represent visual knowledge and knowledge about natural languages.

16.3.4 Rule-Based Systems (Production Systems)

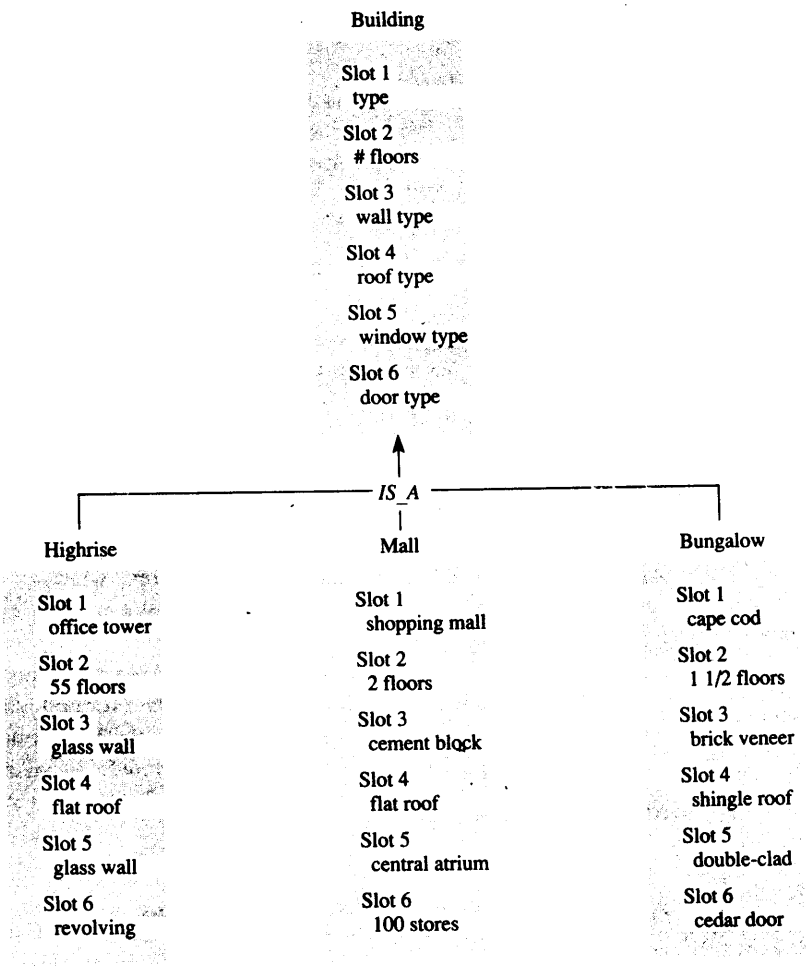
The basic idea in **production systems** is the coupling of a condition with an appropriate action. Each such condition-action pair is called a **rule, production rule,** or simply a **production.** An example of a production is given below:

If condition **then** action

The condition part of a production expresses the conditions under which the rule is valid; the appropriate action to be taken is given by the action part. The action part of the rule changes the state of the system and can introduce new facts. The condition part of the rule is known as an **antecedent** and the action part, the **consequent.** An example of a production rule for the game of hockey, involving a team trailing by one goal in the ultimate minute of the game, can be expressed as a production rule as follows:

If trailing by one goal **and**
remaining time-to-play in game is less than one minute **and**
play is in opponent's zone
then replace goalie by forward.

Figure 16.6 Frame representation of different types of buildings.



A production-based system consists of a set of production rules, a data structure that models the system's current state, and a control subsystem or interpreter that interprets the current state and controls its activity by initiating appropriate action. A rule is said to be **enabled** or **triggered** when the condition part of the rule is satisfied by the current state of the system. An enabled rule is said to be **fired** if the action part of a rule is executed. If the system status is such that more than one rule is triggered, the interpreter may be required to fire one or more of these simultaneously enabled rules; this is referred to as a **conflict resolution.** The conflict resolution can be enacted using its own set of productions. It uses criteria such as priority or ranking, prior selection, arbitrary or random choice, or doing all actions in parallel. The order in which the conditions are examined can be determined a priori or could be adjusted dynamically. The action part can be a single action or a set of procedures that will change the status of the system. The latter change can include disabling a subset of the existing productions and enabling other productions.

An example of a rule-based system is given below. This system examines the causes and the corrective actions to be performed after failing to start a car. The execution of each action will modify the state of the starting system under consideration and if there is more than one problem, all the corrective actions will have to be taken.

if starter cranks the engine very slowly
  then problem may one or more of: extreme cold temperature, battery, cables, connections, voltage regulator, alternator; use jumper cable to start
if starter does not crank but solenoid operates
  then check cables and tighten and clean terminals and check battery voltage
if problem is low battery voltage
  then problem may be battery: check specific gravity and replace if not acceptable;
  problem may be loose, worn, or broken alternator belt: do a visual inspection and if belt is okay, tighten belt, otherwise replace it;
  problem may be cables: visual inspection, clean and tighten connections, and replace broken connectors or cables;
  problem may be voltage regulator: check and replace;
  problem may be alternator: check and repair or replace alternator;
  problem may be shorts in electrical system: locate and correct
if problem loose or worn alternator belt
  then tighten or replace alternator belt
if problem is battery
  then check specific gravity and if acceptable charge otherwise replace battery
if problem is voltage regulator
  then replace voltage regulator
if problem is alternator
  then repair or replace alternator
if battery, cables, and connections are good, solenoid operates, but starter does not crank or cranks slowly
  then replace the starter
if battery, cables, and connections are good but solenoid does not click
  then check ignition switch to solenoid circuit and correct malfunctions
if battery, cables, connections, and ignition switch to solenoid circuit are good but solenoid does not click
  then replace solenoid

Deductive databases are also referred to as logic databases, deductive relational databases, and virtual relational databases. A relational database is a subset of a deductive database.

Up to now we have defined a relation as a set of tuples, i.e., by its extension. The set of tuples can be called the elementary facts, and the relation a base predicate. These sets of facts, which we have referred to as a database, are essentially an extensional database. In the extensional database we define the set of tuples that satisfy a relation. We can also define a relation intensionally by specifying some set of rules. These rules, defining the intensional database, are expressed as well-formed formulas in first-order logic. We can thus consider a database as consisting of a set of rules (or laws) and a set of tuples. The intensional database supplements the extensional one with rules that allow other facts to be derived from those explicitly stored in the extensional database.[5]

Example 16.2

Let the extensional database consist of a parent relation, i.e., a tuple (or fact) of the parent relation tells us the name of the parent of some person. If we also need grandparent names, we can either store the name of the grandparents or—from our knowledge of who the grandparents are—write a derivation rule: The grandparent of X is the person Y in that Z is the parent of X and Y is the parent of Z. ■

If we wanted to find the ancestors or cousins of a person we can specify these as rules. Obviously we save considerably on storage, but more importantly we increase the usability of our database.

Consider the relation PARENT(X,Y) given in Figure 16.4. It represents the fact that Y is a parent of X and is in the extensional database. To find the descendants of an individual, we have to specify a number of rules. The descendants can be specified as follows:

X is a descendant of Y if Y is a parent of X

X is a descendant of Y can be represented as DESC(X,Y), which can be interpreted as a relation DESC having two attributes X and Y. We can write this rule as an implication:

$$\text{PARENT}(X,Y) \rightarrow \text{DESC}(X,Y)$$

[5]We can go even further and consider a database as not having an extension but as consisting entirely of axioms. The extension counterpart could be a set of particularization axioms (to specify the CWA, UNA, and DCA) (Gall 84), (Reit 84).

Databases have been characterized by two basic approaches, the *model-theoretic view* (MTV) and the *proof-theoretic view* (PTV). In the MTV, the database is a model of a first-order theory and queries and integrity constraints are formulas to be evaluated on the model using the semantics of truth. *Model* here is in terms of some set of axioms (in the form of integrity constraints) and an interpretation that makes these axioms true. Queries are evaluated in the MTV under the CWA, UNA, and DCA. In the PTV, the database is a first-order theory (i.e., we try to spell everything out with formulas) and integrity constraints and queries are theorems to be proved. One difference between the MTV and the PTV is that with the former we can add data (a tuple) to the database and still have a model of the same theory (i.e., the model does not have to be changed) while with the latter a different theory would result.

The PTV is a formalization of the concept of the deductive database and not intended to be directly used as the basis of a DBMS implementation.

The above rule gives us the immediate descendants. However, we can find other descendants using a recursive rule:

If Z is a parent of X and a descendant of Y, then X is a descendant of Y

This rule can be expressed as

$$\text{PARENT}(X,Z) \wedge \text{DESC}(Z,Y) \rightarrow \text{DESC}(X,Y)$$

The PARENT relation and the two rules can be used to derive the DESC relation and answer queries such as finding all the descendants of an individual.

Example 16.3

Consider the PARENT relation of Figure 16.4. We can find all the descendants using the above two rules. Initially the DESC relation is empty. We apply the first rule, PARENT(X,Y) → DESC (X,Y), and get the DESC relation (Figure A), which is the same as the PARENT relation.

Figure A DESC relation after the application of the first rule.

DESC

X	Y
Roy	Frank
Jerry	Frank
Myrna	Ruth
Roy	Ruth
Lynn	Roy
Lynn	Rachel
Justin	Lynn
Janet	Myrna
Drew	Sheila
Pavan	Sheila
Sheila	Frank
Frank	George

Now we apply the second rule. This involves the natural join of PARENT (X,Y) and DESC(Y,Z) followed by a projection on the attributes XZ. The new tuples, shown in Figure B, are generated for the DESC relations as a result of the join. (Note: we are renaming the variables in the figure.)

We repeat this step of applying the second rule until no new tuples are added to DESC. The new tuples generated after each application are shown in Figures C and D. No new tuples are generated after the third application of the second rule, so the resulting DESC relation gives all the descendants of a person.

To answer a query such as "Find all the descendants of George," we do a selection on the DESC(X,Y), relation with the following query:

$$\pi_X(\sigma_{Y=\text{George}}\text{DESC})$$

To find all products and their constituent products (at the lowest level), we require the following additional rule. This rule is recursive:

$$\text{PRODUCT}(P,C) \wedge \text{CONSTITUENT}(C,S) \rightarrow \text{CONSTITUENT}(P,S)$$

Here we are defining the rule that an object S is a constituent of an object P if it is a subproduct of P, or it is a constitutent of an object C, which is a subproduct of P. ■

Employees who work together can be derived as follows:

Example 16.5 Consider the relation ASSIGNED_TO *(Prod#,Emp#)*, which gives the employees assigned to a given project. We can find all employees who have worked together on a project by using the following rule:

$$\text{ASSIGNED\_TO}(P,E_1) \wedge \text{ASSIGNED\_TO}(P,E_2) \rightarrow \text{TOGETHER}(E_1,E_2)$$ ■

We can see that the above rules in the form of logic expressions allow us to express recursive queries. This adds to the power of database querying as well as specifying the intensional database.

If $P_1, \ldots, P_n$ and Q are atoms, then $\neg P_1 \vee \ldots \vee \neg P_n \vee Q$ (with a maximum of one unnegated atom) is a Horn clause. The Horn clause with one positive atom is said to contain one conclusion. The conclusion is also known as the head. The atoms $P_1, \ldots, P_n$, specify the conditions to be satisfied and are known as the body of the clause. A Horn clause with no positive atom has no conclusions. A Horn clause with no head may be thought of as integrity constraints, i.e., $P_1 \wedge \ldots \wedge P_n \rightarrow \bullet$ can be interpreted as: (P_1 and P_2 and $\ldots$ P_n) is a violation of an integrity constraint. For example, no individual can be both a father and a mother nor a brother and sister of another individual. This integrity constraint may be specified as:

$$\text{brother}(x,y) \wedge \text{sister}(x,y) \rightarrow \bullet$$

Horn clauses can be expressed easily in PROLOG. If the conditions $P_1, \ldots, P_n$, imply more than one conclusion, i.e., Q is of the form $Q_1 \vee \ldots \vee Q_m$, we write these as m Horn clauses.

In this section we have introduced a powerful extension to the relational database model. Coverage in greater depth is beyond the scope of this text. We give references to relevant literature in the bibliographic notes.

16.5 Expert Systems

Expert systems, also called knowledge base systems, are computer systems designed to implement the knowledge and reasoning used by experts in a particular domain to solve problems in that domain. Knowledge in these systems is obtained from interviews with human experts and represents known procedures, usual practice, heuris-

tics, and rules of thumb. This knowledge is usually implemented as a set of rules, similar to those given in the car-starting example. These computer systems, as do the human experts, use logical inference procedures and compiled production rules (rules of thumb). The explicit domain knowledge, the so-called **institutional memory**, is accessible by the expert system and along with some form of reasoning gives it artificial intelligence. Unlike a human expert, this codified knowledge is of a more permanent nature.

The structure of an expert system, which is built around an appropriate representation of the domain knowledge of an expert, is shown in Figure 16.8. Many expert systems use productions or rules to represent the domain knowledge. The inference system uses the knowledge and applies inference procedures to infer facts not explicitly represented in the knowledge base to solve problems posed by the user. The inference system, in addition, provides the user with the steps used in the reasoning procedure to arrive at a solution to the problem. The user interface is responsible for presenting the user with an easy-to-use interface, and generates responses and understandable explanations to the queries posed by the user.

Abductive reasoning is used in expert systems for applications in areas such as medical or fault diagnostics. Medical diagnostics determines the likely cause for a patient's symptoms. The diagnosis may be multiple, there being a certain level of confidence associated with each possible diagnostic and each level having associated with it a subset of symptoms. Human judgment, along with suggested additional tests, may be required to confirm or rule out some of these multiple diagnoses. For instance, when a starting problem is encountered with the starter cranking the engine very slowly, the diagnosis is that there is a problem with one or more of the following components: extremely cold temperatures, alternator, battery, belt, cables, connections, fuse link, or regulator. Further tests in the form of visual inspection, specific gravity tests, battery voltage, voltage across the battery while the engine is running, or output current from the alternator are required to make a final diagnosis of the problem. In a rule-based expert system, the current known status of the system is matched with the rules and the actions corresponding to one or more of the matched rules are executed, i.e., the rules are fired. As a result of the firing, the state of the system changes.

However, not all expert systems deal with multiple answers or uncertainty. Production or rule-based systems can be deductive systems. Such is the case when the

Figure 16.8 An expert system.

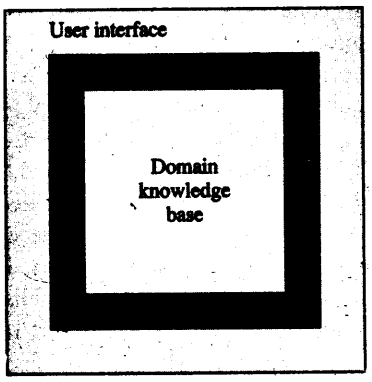
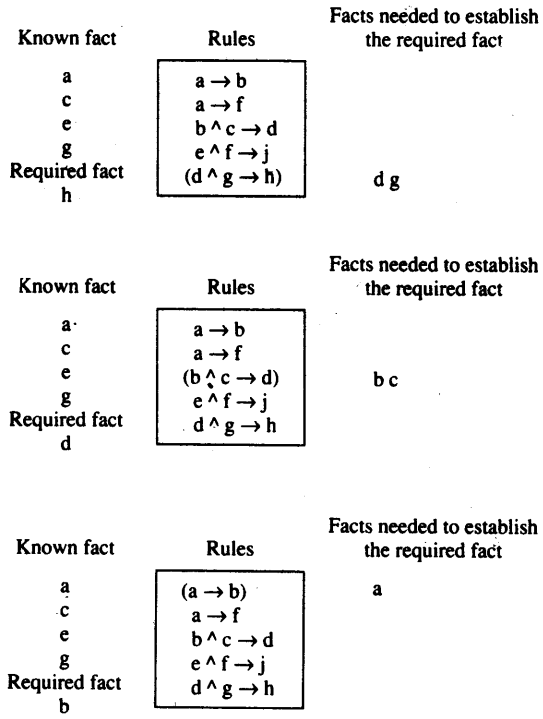


Figure 16.10 Backward chaining.

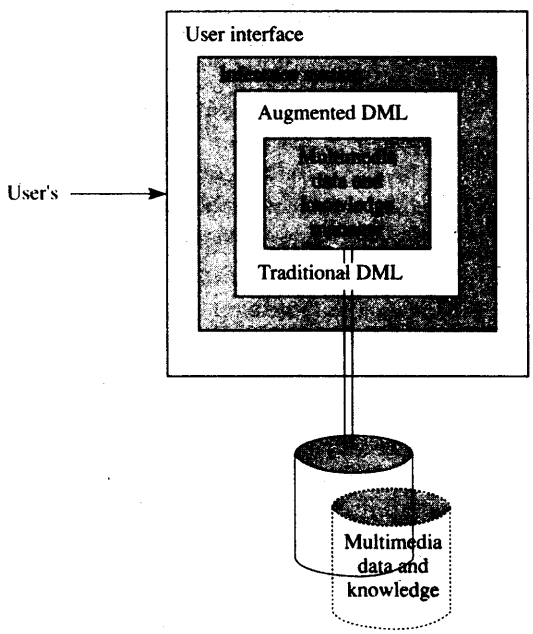


16.6 Expert Database Systems: Integration of Expert Systems in Database Applications

Expert systems have been developed as stand-alone systems. A stand-alone expert system may be required to access data from a database as an ordinary application program. With an integrated approach, the expert system is integrated with the DBMS, as shown in Figure 16.11. In addition to traditional data, the system handles textual and graphical data as well as knowledge. (It must be pointed out that no such integrated system exists to date.) Such an integrated system will be called upon to perform the traditional DBMS functions and use the inference system in aspects of abductive, inductive, and deductive reasoning. The integrated system needs distribution and concurrent access, and at the same time provides enhanced integrity, security, and reliability.

There are obvious advantages in bringing rule-based knowledge representation and reasoning capability to database applications and traditional data processing tasks. The database can be used to store the known facts about objects and events as well as the rules required by the expert system. An ordinary database query not requiring any inference system service could be handled more efficiently by the traditional DML and database manager component of the multimedia database and

Figure 16.11 Integrated expert database system.



knowledge base system; the user interface can forward such queries directly to this subsystem.

The expert system component of such an integrated expert database system can be used to provide a means for interpreting the responses to queries, including responses that contain, for instance, null values. It can be used with appropriate knowledge to enhance the enforcement of integrity and security of the entire system.

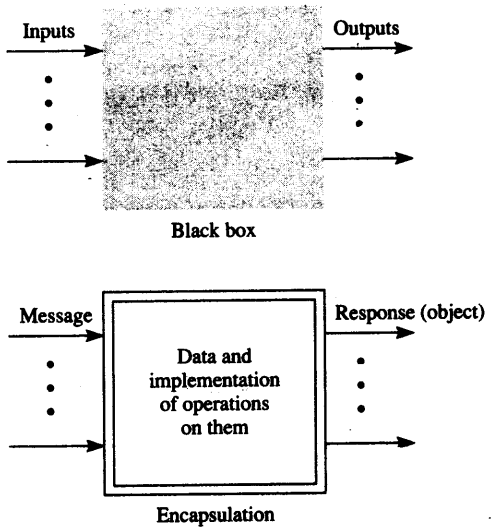
16.7 Object Approach

In the object approach physical entities or abstract concepts of the real world are represented by objects. Objects are distinguished by identifiers and they encapsulate the characteristics or properties of the real world objects as well as their valid operations. The main difference between objects used in object-oriented programming and those in object-database is the persistence. Objects in object-oriented programming persist only for the duration of the program while those in object-database are of a more permanent nature.

It has been predicted that object-oriented programming will be the accepted software development approach of the '90s just as structured programming was the style used in the '70s. One of the results of the evolution of structured programming was the strongly typed requirement made popular by Pascal and the top-down modular approach.

Everyone has a different conception of **object-oriented programming (OOP)** and the **object model (OM).** One yardstick (meterstick!) used by almost everyone is

Figure 16.13 Similarity of a black box and an object.



The messages that an object understands depends on the nature of the object. Objects for numbers understand a message that requests computation and reporting the result. An object representing a thesaurus would understand a message to provide the synonyms, antonyms, or homonyms for an entry in the thesaurus object.

The private memory of an object, which records the value of the data associated with an object, is made available to other objects via the response generated by the object. There is no way to open up an object and look inside it unless the object. via its behavior to messages accepted by it, allows such persual.

Objects are uniform in the method used for communicating, which is by message passing, and because no object is given special status. There is no distinction between objects supplied by the system and those created by a user.

The basic problem in OOP is to determine the kinds of objects that should be implemented. In addition, for each object, we have to determine the messages it will accept and the response provided by such messages. The choice of object depends on the application and use of the system. Sending a message in OOP is equivalent to calling a procedure in procedural language or providing inputs to a black box (see Figure 16.13).

16.7.2 Names and Identity

What's in a name?[6]

> Identity is such a simple and fundamental idea that it is hard to explain otherwise than through mere synonyms. To say that X and Y are identical is to say that they are the

[6]William Shakespeare, Romeo and Juliet, Act II scene 2.