Key Terms

database machines
backend software
backend computer
associative memory
intelligent memory
processor per track
cellular logic device
processor per surface
processor per device
multiprocessor and cache
special hardware

Bibliographic Notes

With the development of database management systems, the load placed on the system of second and third generation computers far exceeded their capabilities, which led to the concept of database computers. One of the first reports of a prototype development of backend database computers was the XDMS project of Bell Labs (Cana 74). Earlier, Slotnick (Slot 70) had proposed a logic-per-track storage device. The cellular logic device is a generalization of Slotnick's logic-per-track concept. Examples of this approach are the CASSM (Su 79), the RAP (Schu 78), and the RARES (Lin 76) projects. The processor-per-surface approach was used in the DBC project (Bane 78). DBC/1012 (Tera a) is an example of the processor-per-surface approach which, with the distribution of data on different AMPs, achieves the efficiency of the MPC approach. The DIRECT project (Dewi 81) is another example of the MPC approach. More recent systems are described in (Hsia 83), (MDBS); (Fish 84), (Jasmin); (Kits 85), (Grace); and (Dewi 86), (GAMMA).

(Mary 80) presents a tutorial on the software backend computer approach. With the increasing use of the relational model, there was an increase in emphasis on developing systems to improve the performance of the relational model in hardware (Babb 79), (Bane 78), (Dewi 81), (Lin 76), (Smit 79). The use of content-addressable memories is not cost effective and they remain controversial (Hawt 81). Commercial database machines continue to use conventional rotating memories.

Textbook-oriented discussions of database computers are presented in (Hsia 83) and (Su 88).

Bibliography

(Babb 79) E. Babb, "Implementing a Relational Database by Means of a Specialized Hardware," *ACM Trans. on Database Systems* 4(1), March 1979, pp. 1–29.

(Bane 78) J. Banerjee, D. K. Hsiao, & R. I. Baum, "Concepts and Capabilities of a Database Computer," *ACM Trans. on Database Systems* 3(4), 1978, pp. 347–384.

(Berr 79) P. B. Berra & E. Oliver, "The Role of the Associative Array Processor in Database Machine Architecture," *Computers* 12(3), March 1979, pp. 53–63.

(Cana 74) R. H. Canady, R. D. Harrison, E. L. Ivie, J. L. Ryder, I. A. Wehr, "A Backend Computer for Database Management," *Comm. of ACM* 17(10), October 1974, pp. 575–582.

(Date 83) C. J. Date, *An Introduction to Database systems*, vol. 2. Reading, Ma: Addison-Wesley, 1983.

(Dewi 81) D. J. DeWitt, "Direct—A Multiprocessor Organization for Supporting the Relational Database Management Systems," *IEEE Trans. on Computers* C–28, June 1979, pp. 395–406.

(Dewi 86) D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, & M. Muralikrishna, "GAMMA: A Performance Dataflow Database Machine," Proc. of the 12th International Conf. On Very Large Data Bases, Kyoto, Japan, August 1986, pp. 315–344.

17.3.2 System Facilities of the DBC/1012

The DBC/1012 provides a number of system facilities for database security, integrity, and concurrency control.

Security is implemented by means of a session protocol. A user is required to log on to the DBC/1012 to establish a session. The logon procedure identifies the user (or application program) and provides an account number and a password. A session is established once the logon parameters are accepted. A session ends when the user logs off. Unauthorized access or an attempt to access outside a session is denied and appropriately reported.

Concurrency is implemented by locking. The locking granularity could be the entire database, a relation, or a tuple. There are four modes of locking provided by DBC/1012: exclusive, write, read, and access. The access lock can be used by users who are not concerned with data consistency. The degree of concurrency is increased since the access lock allows read operations to be executed simultaneously against a data item locked in the write mode.

Recovery is implemented by the use of transient and permanent journals. The transient journal is a log of updates to the database. The log entry consist of the transaction identification and the before image or the modified data items. The transient journal is used to undo a single transaction error.

The permanent journal is an optional second method of recovery implemented in the DBC/1012. The DBA decides to log either the before image or the after image of data items in the log of the permanent journal. The log could be single or double; in the latter case redundancy in the log is provided by recording two copies of the before or after image.

Archiving (dump) is performed by making copies of the database and permanent journal at regular intervals. Checkpoint facility is part of the permanent journaling feature.

Recovery from failures is achieved by rollback or roll forward optionally to a specified checkpoint.

17.4 Summary

In the traditional approach to database systems, the DBMS runs on the same computer as the user programs. The data in this approach is stored on conventional rotating memories. It is necessary to move the data to the central processing unit for processing and to determine what portion of it is needed to respond to a user's query. We discussed a number of approaches that have been used to place some of the database management load on a separate system. In some of these approaches, some form of computing capability is provided near the data, which avoids moving superfluous data to the main processing unit. The processor per track or cellular logic, the processor per surface, the processor per device, and the multiprocessors and cache are attempts to provide processing close to the data. We also described one instance of a commercially available special-purpose computer that handles the database management functions.

owner and creator of these relations, indexes to be maintained, a list of authorized users and their access rights. The scheme is maintained in the DD/D.

17.3.1 Operation of the DBC/1012

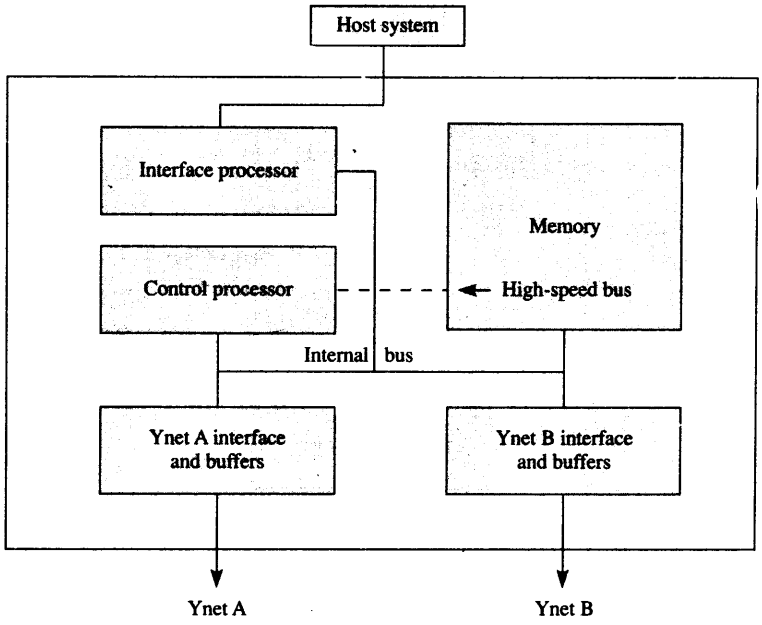
The database system in the DBC/1012 consists of the following components: session control, dispatcher, and database manager. Each IFP and COP is responsible for the first two components and in addition these processors interface with the host systems. The database manager is implemented on each AMP and is responsible for providing the transformation from the logical database organization to the physical level; the data is stored on the DSUs. The access aid used by the DBM is a two-level index consisting of a master index and a cylinder index; binary search is used on these indexes.

The user defines and manipulates the database using the following facilities:

- In DBC/SQL, the Teradata query language. This is the facility used to define and manipulate the database. Thus, the user can define relations or views on existing relations as well modify them using statements in DBC/SQL. Statements in DBC/SQL allow the user to control access to the database by establishing users and their access profiles.
- Interactively, by statements in the Interactive TEradata Query (ITEQ) language. This includes functions for retrieving metadata about the database; entering, editing, and executing DBC/SQL statements; and specifying the format of the output.
- In a batch mode, using a facility provided by the Batch TEradata Query (BTEQ) language, wherein a number of DBC/SQL statements along with BTEQ batch commands can be executed.
- By DBC/SQL statements included in application programs in a high-level language. These statements are converted by a language preprocessor into calls to CLI routines. After the compilation of the source program, these CLI routines are link-edited with the object code to generate a load module ready for execution. It is also possible to dynamically load the CLI routines at run time. At execution time, the CLI service routines generate a query request, which is communicated by a UTC routine to the TDP.
- In a natural query language such as INTELLECT or a fourth-generation language such as NOMAD.
- By using calls to CLI routines in a high-level language.
- By using a data directory/dictionary facility to access the meta information regarding the database objects.

The user's query requests are communicated to the TDP via UTC routines by the CLI service routines. The TDP is responsible for managing the communication between the application program or the user and the DBC/1012. On receiving a query request, the TDP creates a message for the IFP, which is communicated via the host to DBC/1012 interface. The CLI routines are also responsible for receiving the response to the DBC/SQL statements from the DBC, via the TDP, and forwarding it to the application program or user that originated the request.

Figure 17.10 Interface processor (adapted from [Tera a]).



Session control involves processing the logon and logoff requests from the host. The messages to and from the TDP in the host are under the control of the host interface.

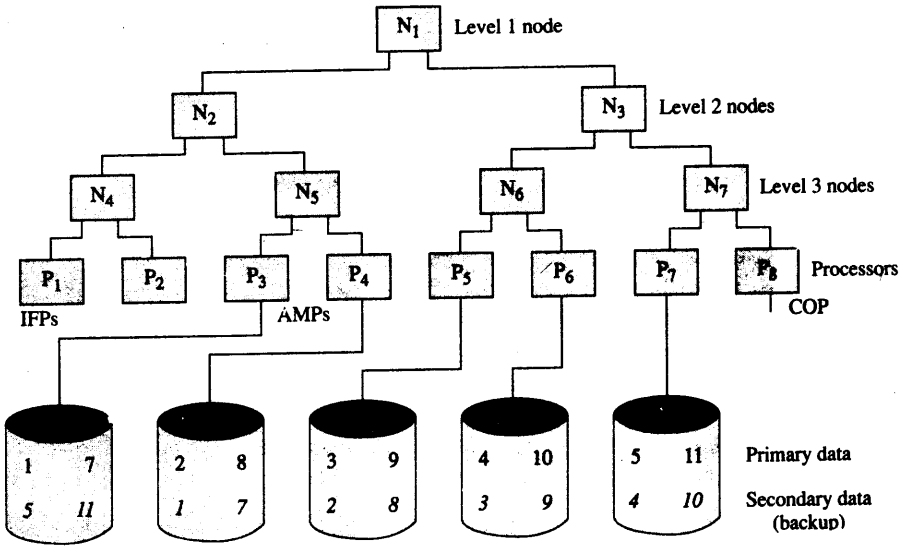
A DBC/SQL request from the host is semantically interpreted by the parser. This interpretation may need a reference to the system information stored in the data dictionary/directory to resolve symbolic references and determine integrity constraints. The parser generates a number of work steps required to process the request and sends these to the dispatcher.

The dispatcher controls the execution of work steps and also performs the assembly of the response to be sent to the host via a response control subsystem. The dispatcher schedules the execution of these work steps and passes them to the Ynet interface, which in turn sends them over the Ynet to one or more AMPs. The dispatcher is also responsible for monitoring the status of the work steps in the AMPS and interacting with the response control. The response control is responsible for the assembly and transmission of the response for a request from the host.

AMPs and DSUs

The access module processor (Figure 17.11) is very similar to the IFP and uses some of the same components. The AMPs receive requests for database access over the Ynet and respond by sending the required information back to the requesting IFP or COP over the Ynet. Each AMP is connected to both Ynets and could have a maximum of two data storage units. The database manager (DBM) subsystem is resident on each AMP in the DBC/1012 and is responsible for executing the functions of

Figure 17.9 Basic configuration of the Ynet (adapted from [Tera a]).



number of work steps to respond to this request. These work steps are encoded into data blocks that travel up the hierarchy from the IFP processor level to the node at level 3, and then down to one or more AMPs. The downward transmission uses a broadcast mode; the upward transmission is controlled by control information associated with the data block. Data retrieved by the AMPs travels up the network. Contention logic in the network is used to sort the data moving up from the AMPs. The control information in the data block is also used to sequence the arrival of blocks from AMPs to a given IFP in a certain order and achieve merge/sorting of the relevant data.

The following types of communication are provided by the Ynet: between any two processors; from one processor to a group of processors; from a group of processors to a single processor; or between processors to synchronize their operations on data. A Ynet can be expanded to support up to 968 processors.

IFP

The IFP (Figure 17.10) interfaces both with the host and the Ynets and manages the traffic between the two. The number of IFPs depends on this traffic and can be adjusted to match it. Each IFP is connected to both Ynets. The functions implemented in the IFPs are the following: host interface, session control, parser, dispatcher, and Ynet interface. These functions are implemented in hardware or software and are briefly described below.

The Ynet interface in the IFP controls the transmission of messages to and the receipt of responses from the AMPS. A message may be transmitted to a single AMP or to a group of them

Figure 17.6 Moving head disk with a single read/write head and processor per surface.

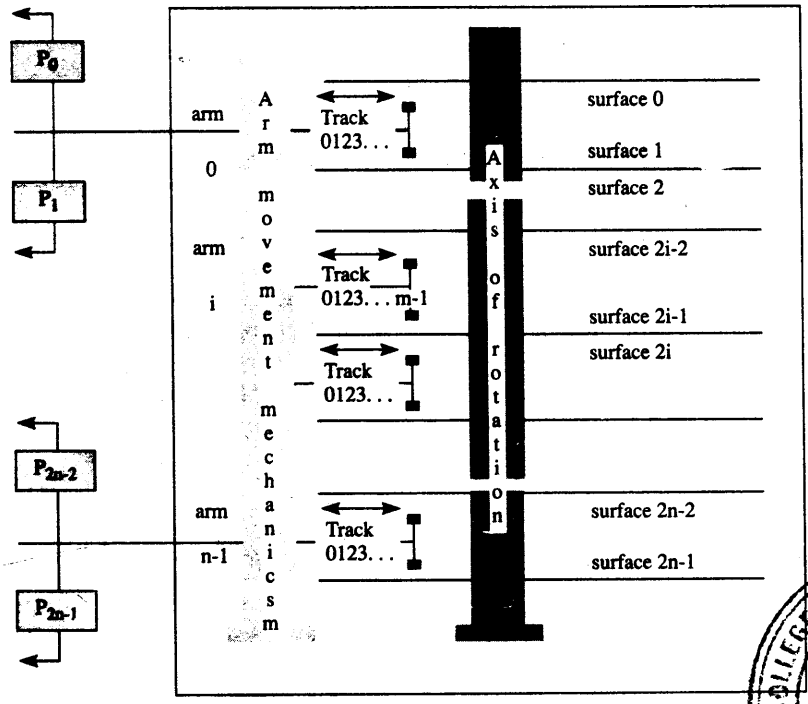


Figure 17.7 Multiprocessor and cache system.

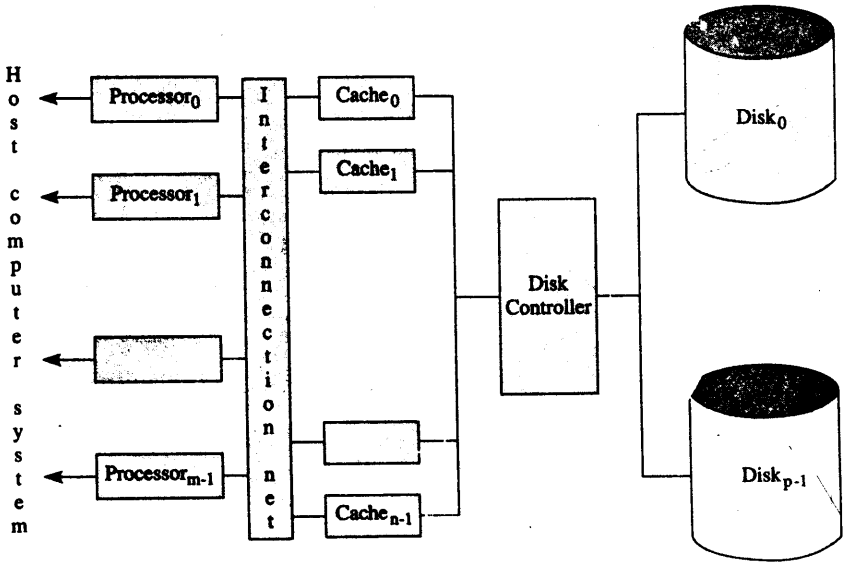
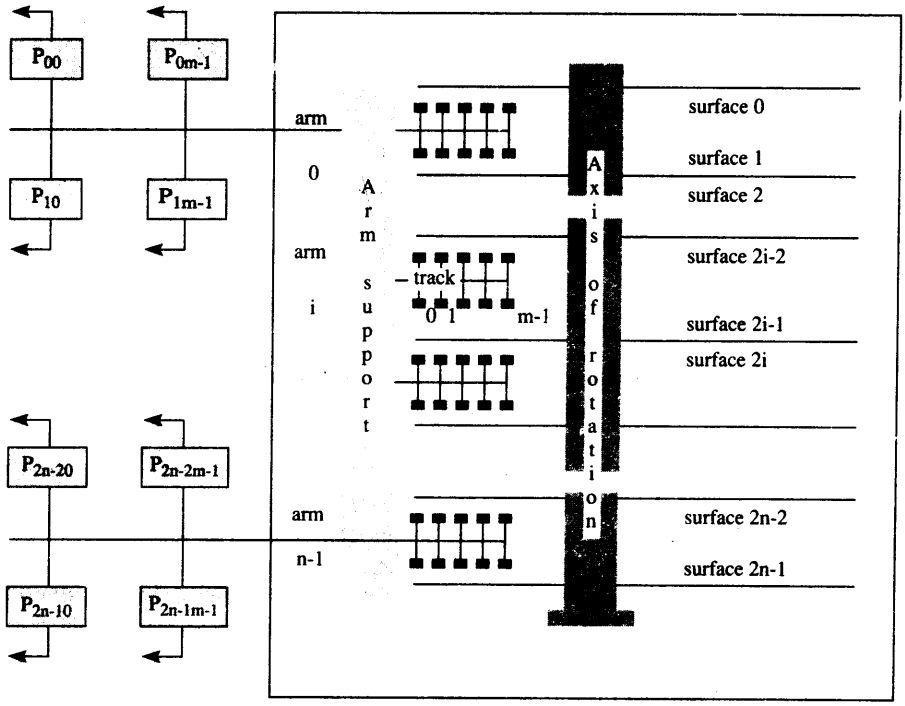


Figure 17.5 Fixed-head disk with read/write head and processor per track.



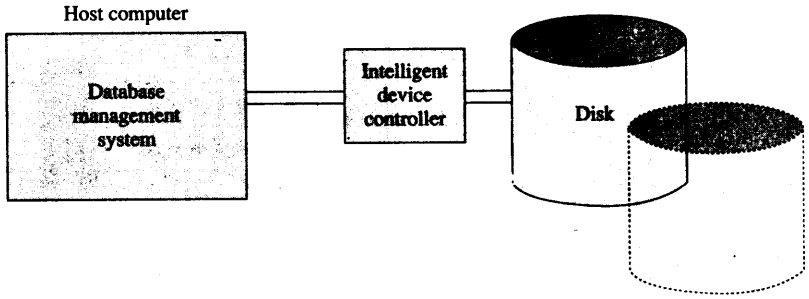
The **processor-per-device** approach is an attempt to further reduce the number of processing elements associated with the storage, and hence the cost. In this scheme there is a single processor associated with each storage device. The processor acts as a filter between the host computer and the device. Indexes are required to reduce the number of passes and the amount of data actually processed by this filter processor.

The **multiprocessor and cache** scheme (Figure 17.7) is an attempt to optimize the cost-performance factor by allowing the filter processors to be assigned to process the data from any one of a number of storage devices, or from a number of different tracks or cells of a single device. The data to be processed is placed in one of the n high-speed memory caches, there being m filter processors. The interconnection network is used to connect any one of the m processors to any of the n caches. Up to m caches can be processed simultaneously; the data in these caches could be from distinct devices or from the same device. With $n > m$, some of the empty caches could be filled while m caches are being processed and buffer the difference between the processing rate and the device access rate.

17.2.3 Special Hardware Approach

In the **special hardware** approach, instead of using a conventional computer as the engine in the backend for running the database management system, a specially de-

Figure 17.4 Associative memory approach.



17.1, running the application as well as the DBMS, or it could be a dedicated back-end computer.

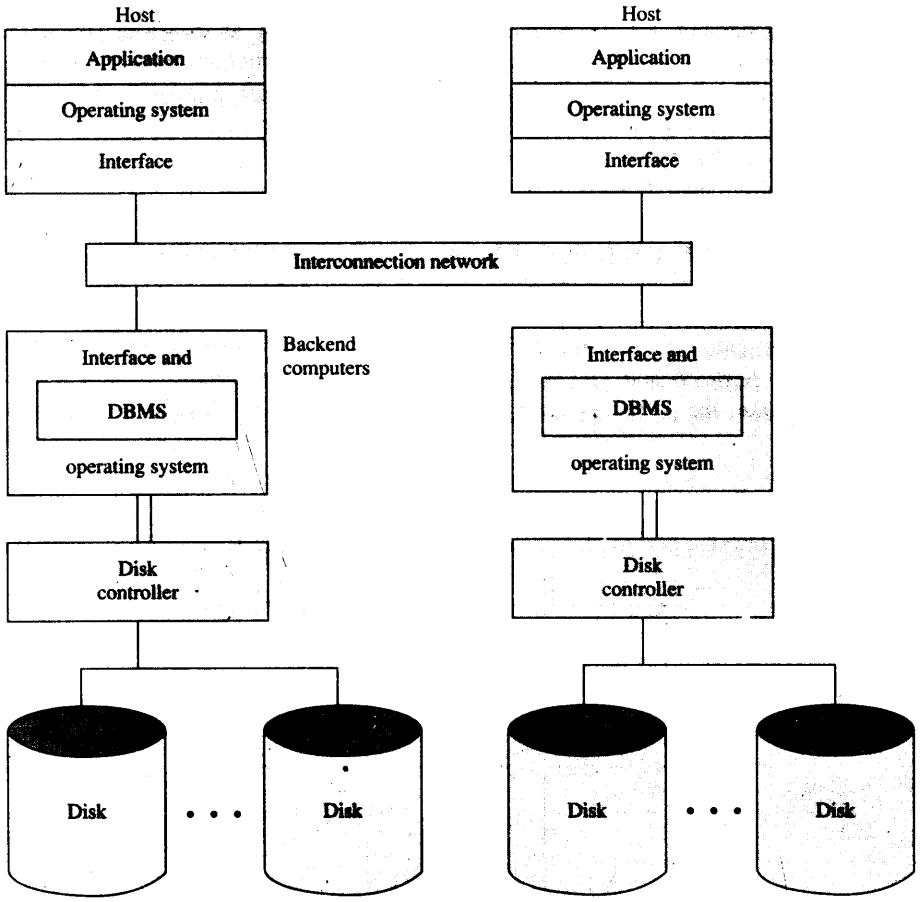
The processing capability associated with the secondary memory is provided by VLSI-based microprocessors and hence is cost effective. The storage device could be electromechanical in the form of rotating disks or drums, or it could be nonmechanical, e.g., magnetic bubble memories or charge-coupled devices. (In the following discussions, we assume that the storage devices are electromechanical; however, the same concepts can be applied to the nonmechanical devices.) The processing capability may be associated with a storage device in one of the following manners: processor per track of a fixed-head type storage device; processor per surface of a moving-head type storage device; processor per storage device; or multiprocessor and cache approach.

In the **processor-per-track** approach, a processor is associated with each track of the secondary storage device, the latter being a fixed-head disk or drum or other such device (Figure 17.5). This type of structure is also called a **cellular logic device,** since logic is associated with each cell of memory. Data from the track is processed by the associated processor and data from all tracks can be processed simultaneously. Thus, the entire contents of the storage device can be processed in one pass, which in the case of a rotating storage device is a single revolution. Since all data can be processed in a single pass, indexes are not needed.

The disadvantage of the processor-per-track scheme is that the data from all tracks of a single device is not necessarily required and the concurrent processing of the irrelevant data is unproductive. The cost of this type of storage device is high. However, with the ultralarge-scale integration (ULSI) of logic components, the cost is expected to decrease.

The **processor-per-surface** method is an attempt to associate processing power with each read/write head of a moving-head type secondary storage device (Figure 17.6). The amount of data that can be processed per pass by each processor is the same as in the processor-per-track approach; however, to process all the data from the storage device would take m passes or revolutions, where m is the number of tracks per surface. In the case of mechanical devices such as disks and drums, the movement of the head from track to track takes a finite amount of time and this will have to be accounted for in the total time required to process the data from the device. If the storage device is nonmechanical, the switching of the cells to be processed can be done at much faster electronic rates. To reduce the number of passes, indexes are necessary for these storage devices.

Figure 17.3 Multiple backend computers serving multiple hosts.



tribution of data on the multiple backend computers; the maintenance of the directory containing this information about the data distribution; if such a directory is not maintained, then the overhead for determining the location of required data; consistency enforcement if data is replicated.

17.2.2 Processor Associated with Memory or Intelligent Memory Approach

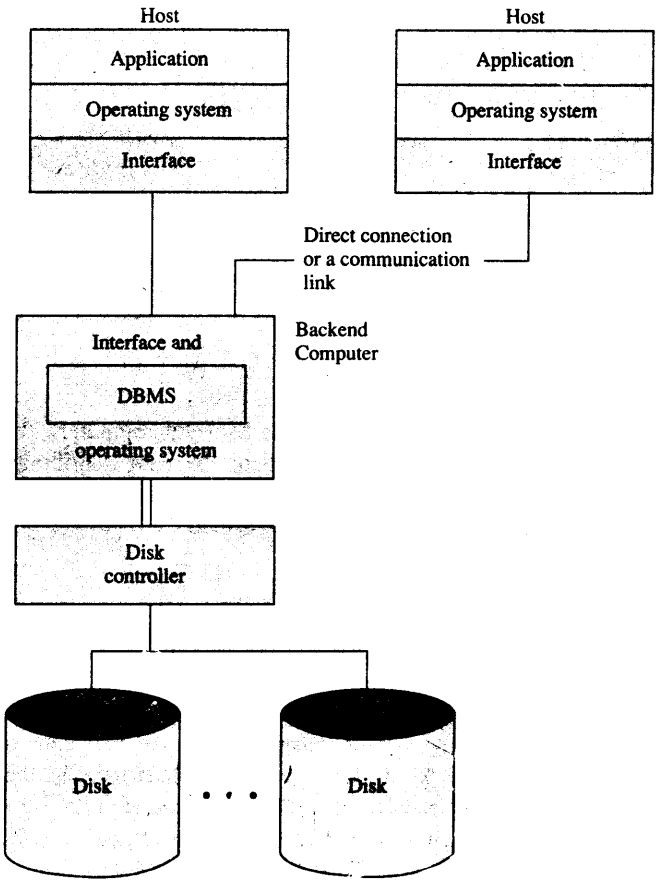
In the **intelligent memory** approach (see Figure 17.4), sufficient processing logic is associated with the secondary memory so that data can be processed before being transmitted to the host processor. The host runs the database management system. If sufficient processing capability is associated with the secondary storage device controller, it can intercept data from the secondary storage device to determine its usefulness. There is no need to move superfluous data to the host system running the database management system. The host could be a conventional system as in Figure

cation programs. However, since the overall system has more components than a conventional system, the likelihood of failure is increased.

A dedicated database machine can be used to support the database operations for a number of host computers and/or workstations, as shown in Figure 17.2. Such an approach, where a number of hosts share one or more backend computers, permits cost-effective sharing of both the data and the database management functions. However, unless the dedicated database machine has the required capacity to handle this load, it will create a bottleneck. Furthermore, failure of this dedicated system would bring the operations of these hosts to a halt.

An alternative solution to relieve the bottleneck and to increase the reliability of the backend system is to incorporate multiple dedicated database machines in the database management functions, as shown in Figure 17.3. In this variation of the backend approach, a number of backend computers can be used to handle a very large database, the latter being distributed to optimize performance by allowing parallel retrieval and processing of data required simultaneously. However, with this scheme, the problems encountered are the following: the need to determine the dis-

Figure 17.2 Backend database computer approach.



attempted. The references to some of these systems are given in the bibliographic notes. These approaches can be classified as one of the following: backend software approach; processor associated with memory or intelligent memory approach; special hardware approach. We briefly describe these approaches in the following sections.

17.2.1 Backend Software Approach

In the **backend software** approach, sometimes called the **backend computer** approach, the host computer where the applications are located is attached to a dedicated general-purpose computer and a conventional database management system runs on this backend computer. This dedicated backend computer is responsible for carrying out the database functions of locating and retrieving the required data as well as ensuring security, enforcing consistency criteria, and providing for recovery operations. This releases the host computer from database management functions. Superior performance can be achieved by parallel processing of the application programs and the database operations in distinct processors. A single backend computer can be attached to a single host or a number of hosts, not necessarily identical, can share a single backend computer.

A database request from an application program in the host computer is intercepted by special interface software, which sends the request to the dedicated backend database machine. The backend machine performs the required data access and processing operations to derive the response for the request and this response is sent back to the host.

The backend machine can be a conventional computer dedicated to running a conventional database management system. It can also be a system consisting of one or more specialized processors using traditional secondary storage devices or associative memories of one or more types. **Associative memory** has logic associated with each word or each bit of the memory. The logic is used to simultaneously examine the contents of the entire memory. Matching words are flagged and could be rapidly located for subsequent processing.

Regardless of the nature of the backend system, it is dedicated to performing the database functions in an optimal manner to achieve cost-effective performance. Higher performance is achieved by the parallelism inherent in such a system.

There are certain advantages and disadvantages in dedicating a separate system for the database functions. We already mentioned the higher performance attainable with such a system as a consequence of parallelism and specialization. The performance here is measured in terms of the overall system throughput and not necessarily the response for a single query. The response to a query in a backend approach involves an overhead in the form of communication between the host and the backend computers. As a result, the response time for a query is likely to be worse in the backend approach compared to the conventional approach where communication between computers is not required.

In the backend approach, since the data is under the control of a dedicated system, data security is enhanced. This is because no user has direct access to the backend system, all requests being handled through the host interface. Also, since no application programs run on the dedicated system, the reliability of the database system is improved; there is freedom from crashes that occur due to incorrect appli-

In this chapter we discuss a number of approaches used to relieve the main computer system of the burden of running the database management system and to handle the superfluous data not required for deriving the response to a user's query.

17.1 Introduction

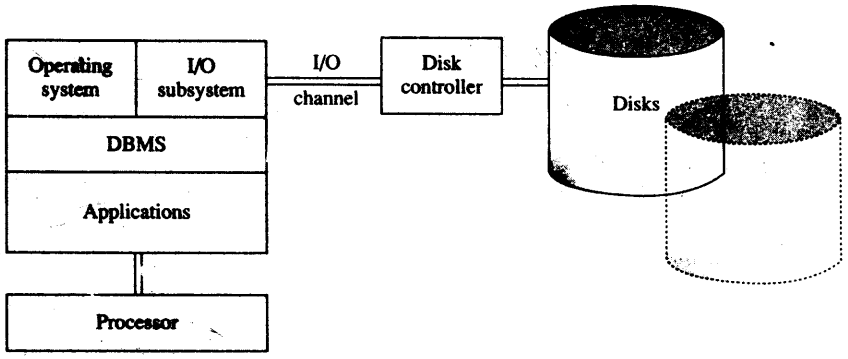
In the traditional approach to database systems (see Figure 17.1), the data is stored on secondary storage devices and the ability to perform any logical or arithmetic computing operations is limited to the central processor. Data has to be moved from the secondary storage devices to the main memory attached to the central processor. Once the data is transferred to the main memory, the processor can access it and determine if the data is useful. Thus it is likely that a large quantity of superfluous data will also be retrieved and processed. It has been estimated that on the average, only 10% of the retrieved data is found to be pertinent. The utilization of indexes is one approach used to reduce this wasteful movement and processing of data. However, the indexes themselves take up considerable storage space and generate substantial traffic on the input/output channels as well as a heavy processing load on the processor.

17.2 Database Machine Taxonomy

The approach taken in **database machines** is to offload the database management functions onto a special processor and optionally add some level of computing capability closer to the data. The special processor relieves the main computer system of the task of managing the database; the extra level of computing capability makes it feasible to decide whether a given set of data will be useful in the evaluation of a query without having to transfer the data to a central processing unit.

A number of approaches to moving the computing power closer to the data have been proposed, and experimental systems for some of these proposals have been

Figure 17.1 Conventional approach.



Chapter

17

Database Machines

Contents

(East 86) G. M. Eastman, "Three Uses of Object-Oriented Databases to Model Engineering Systems," Proceedings, International Workshop on Object-Oriented Database Systems, Sept. 1986, pp. 215–216.

(Find 79) N. V. Findler, ed., *Associative Networks: Representation and Use of Knowledge by Computers*. New York: Academic Press, 1979.

(Fish 87) D. H. Fishman, et al., "Overview of the IRIS DBMS," *ACM TOOIS* 5(1), January 1987, pp. 48–69.

(Fish 88) D. H. Fishman, et al., *Overview of the IRIS DBMS*. Palo Alta, CA: 1988. H–P Labs, 1988.

(Fros 86) R. Frost, *Introduction to Knowledge Base Systems*. New York: Macmillan, 1986.

(Gall 78) H. Gallaire & J. Minker, *Logic and Databases*. New York: Plenum Press, 1978.

(Gall 84) H. Gallaire, J. Minker, & J.–M. Nicolas, "Logic and Databases: A Deductive Approach," *ACM Computing Survey* 16(2), June 1984, pp. 153–185.

(Gold 83) A. Goldberg & D. Robson, *Smalltalk–80: The Language and Its Implementation*. Reading, MA: Addison-Wesley, 1983.

(Goya 87) P. Goyal, T. S. Narayanan, Y. Z. Qu, & F. Sadri, "Requirements for an Object-Based Integrated Systems Environment," Technical Report CSD–87–007, Dept. of Computer Science, Concordia University, 1987.

(Gray 84) P. Gray, *Logic, Algebra and Databases*. Chichester, England: Ellis Horwood, 1984.

(Hamm 81) M. Hammer & D. McLeod, "Database Description in SDM: A Semantic Database Model," *ACM Transactions on Database Systems* 6(3), 1981, pp. 351–386.

(Hard 86) T. Harder, "New Approaches to Object Processing in Engineering Databases," Proceedings, International Workshop on Object-Oriented Database Systems, Sept. 1986, p. 217.

(Huds 86) S. E. Hudson & R. King, "CACTIS: A Database System for Specifying Functionally Defined Data," Proceedings, International Workshop on Object-Oriented Database Systems, Sept. 1986, pp. 26–37.

(Isra 86) D. Israel, "AI Knowledge Bases and Databases," in M. L. Brodie & J. Mylopoulous, eds., *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*. New York: Springer-Verlag, 1986.

(Jone 79a) A. K. Jones, "The Object Model: A Conceptual Tool for Structured Software," in R. Bayer, R. M. Graham, & G. Seagmuller), eds., *Operating Systems: An Advanced Course*. NY: Springer-Verlag, 1979, pp. 8–18.

(Jone 79b) A. K. Jones, R. J. Chandler, I. E. Durham, K. Schwans, & S. Vegdahl, "StarOS: Multiprocessor Operating System for Support of Task Forces," Proceedings 7th ACM Symposium on Operating System Principles, Dec. 1979, pp. 117–129.

(Jones 86) M. B. Jones & R. F. Rashid, "Mach and Matchmaker: Kernel and Language Support for Object-Oriented Distributed Systems," Proceedings, OOPSLA '86, Sept. 1986, pp. 67–77.

(Katz 86) R. H. Katz, E. Chang, & R. Bhateja, "Version Modeling Concepts for Computer-Aided Design Databases," Proceedings, ACM SIGMOD '86, May 1986, pp. 379–386.

(Kers 86) L. Kerschberg, ed., *Expert Database Systems: Proc. from the First International Workshop*. Menlo Park, CA: Benjamin/Cummings, 1986.

(Keta 86) M. A. Ketabchi, "Object-Oriented Data Models and Management of CAD Databases," Proceedings, International Workshop on Object-Oriented Database Systems, Sept. 1986, pp. 223–224.

(Khos 86) S. Khoshafian & G. P. Copeland "Object Identity," Proceedings, OOPSLA '86, Sept. 1986, pp. 406–415.

(Kim 88) W. Kim & F. Lochovsky, *Object-Oriented Concepts and Databases*. Reading, MA: Addison-Wesley 1988.

(Know 83) Special issue on knowledge representation. *IEEE Computer* 16(10), October 1983.

(Korf 66) R. R. Korfhage, *Logic and Algorithms*. New York: John Wiley, 1966.

(Kowa 79) R. Kowalski, *Logic for Problem Solving*. New York: North-Holland, 1979.

(Kuip 75) B. J. Kuipers, "A Frame for Frames: Representing Knowledge for Recognition," in D. G. Bobrow

(Barr 81) A. Barr & E. A. Feigenbaum eds., *The Handbook of Artificial Intelligence*, vol. 1. Los Alatos CA.: William Kaufman, 1981.

(Birt 73) G. M. Birtwhistle, O. J. Dahl, B. Myrhaug, & K. Nygaard, *SIMULA Begin*. Auerbach Publishers, 1973.

(Blac 85) A. P. Black, "Supporting Distributed Applications: Experience with Eden," Proceedings 10th ACM Symposium on Operating System Principles, 1985, pp. 181–193.

(Bobr 75) D. G. Bobrow & A. Collins, eds., *Representation and Understanding: Studies in Cognitive Science*. New York: Academic Press, 1975.

(Brac 83) R. J. Brachman, "What IS–A is and Isn't: An Analysis of Taxonomic Links in Semantic Network," *IEEE Computer* 16(10), 1983.

(Brac 86) R. J. Brachman & H. J. Levesque, "What Makes a Knowledge Base Knowledgeable? A View of Databases from the Knowledge Level," in L. Kerschberg, ed., *Expert Database Systems: Proceedings from the First International Workshop*. Menlo Park, CA: Benjamin/Cummings, 1986, pp. 69–78.

(Brod 84) M. L. Brodie, J. Mylopoulos, & J. W. Schmidt, eds., *On Conceptual Modelling: Perspective from Artificial Intelligence, Databases and Programming Languages*. New York: Springer-Verlag 1986.

(Brod 86a) M. L. Brodie & J. Mylopoulos, eds., *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*. New York: Springer-Verlag, 1986.

(Brod 86b) M. L. Brodie, R. Balzer, G. Wiederhold, R. Brachman, & J. Mylopoulos, "Knowledge Base Management Systems: Discussions from the Working Group" in L. Kerschberg, ed., *Expert Database Systems: Proceedings from the First International Workshop*. Menlo Park, CA: Benjamin/Cummings, 1986, pp. 19–23.

(Brod 86c) M. L. Brodie & J. Mylopoulos, "Knowledge Bases vs. Databases," in M. L. Brodie, & J. Mylopoulos, eds., *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*. New York: Springer-Verlag, 1986, pp. 83–86.

(Card 85) L. Cardelli & P. Wegner, "On Understanding Types, Data Abstraction, and Polymorphism," *Computing Surveys* 17(4), December 1985, pp. 471–522.

(Care 86) M. J. Carey, D. J. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J. E. Richardson, & E. J. Shekita, "The Architecture of the EXODUS Extensible DBMS,' Proceedings, International Workshop on Object-Oriented Database Systems, Sept. 1986, pp. 52–65.

(Chen 76) P. P. Chen, "The Entity-Relationship Model—Towards a Unified View of Data," *ACM Transactions on Database Systems* 1(1), March 1976, pp. 9–36.

(Chri 86) S. Christodoulakis, F. Ho, & M. Theodoridou, "The Multimedia Object Presentation Manager of MINOS: Symmetric Approach," Proceedings, ACM SIGMOD '86, May 1986, pp. 295–310.

(Cope 84) G. Copeland & D. Maier, "Making Smalltalk a Data Base System," Proceedings, ACM SIGMOD '84, June 1984, pp. 316–325.

(Dahl 66) O. J. Dahl & K. Nygaard, "SIMULA, an ALGOL Bases Simulation Language," *Comm. of the ACM* 9(9), September 1966, pp. 671–678.

(Danf 88) S. Danforth, & C. Tomlinson, "Type Theories and Object-Oriented Programming," *ACM Computing Surveys* 20(1), March 1988, pp. 29–72.

(Dasg 85) P. Dasgupta, R. J. LeBlanc, & E. Spafford, "The Clouds Project: Designing and Implementing a Fault Tolerant, Distributed Operating System," Technical Report GIT-ICS 85/28, Georgia Institute of Technology, School of Information and Computer Science, 1985.

(Dasg 86) P. Dasgupta, "A Probe-Based Monitoring Scheme for an Object-Oriented Distributed Operating System," Proceedings, OOPSLA '86, September 1986, pp. 57–66.

(Ditt 86) K. R. Dittrich, "Object-Oriented Database Systems: The Notion and the Issues," Proceedings, International Workshop on Object-Oriented Database Systems, Sept. 1986, pp. 2–4.

(Dixo 89) G. N. Dixon, G. D. Parrington, S. K. Shrivastaua and S. M. Wheater, "The Treatment of Persistent Objects in Arjuna," *The Computer Journal*, 32(4), August 1989, pp. 323–332.

an object is determined by the message to which it responds; this set of operations is called the object's message protocol. Such a set of operations for each message is called a method and determines the response generated by the object. The collection of a group of identical objects into a class allows the sharing of common methods. In the object approach, inheritance is used to allow different objects to share attributes and methods.

Along with the lack of a clear, well-defined and accepted object model there is a lack of uniformity in the concept of an object-oriented database system. Object database can be classed as either an extension to an existing system or as an object-oriented DBMS (OODBMS) wherein the data model supports the object approach.

Key Terms

knowledge base management system (KBMS)
reasoning facility
deductive reasoning
inductive reasoning
abductive reasoning
explanation facility
knowledge representation scheme
exception-handling features
knowledge independence
robust
metaknowledge
semantic network
property inheritance mechanism
override
proposition
propositional logic
propositional calculus
modus ponens
chain rule
reductio ad absurdum
predicate
sorts
function
first-order logic predicate calculus
first-order predicate calculus
Horn clause
closed world assumption (CWA)
unique name assumption (UNA)
domain closure assumption (DCA)
frame
production system
rule
production rule
production
antecedent
consequent
enable
trigger
fire
conflict resolution
forward chaining
backward chaining
procedural representation method
deductive database
extensional database (EDB)
intensional database (IDB)
expert system
institutional memory
object-oriented programming (OOP)
object model (OM)
reusability
black box approach
extendability
compatibility
object-oriented approach (OOA)
object
message
method
class
instance
subclass
message interface
message protocol
addressability
identity
object identifier
class inheritance
multiple inheritance
object independence
partial inheritance
dynamic inheritance

Exercises

16.1 Write the production rules for an expert system to help in advising a client of a bank as to the type of account or accounts he or she should open.

16.2 Using the production rules of Figure E, show the order in which the rules will be fired in

easily enforced. Since the operations allowed on an object are encapsulated, its interactions with other objects are known and hence predictable. This allows ease in extension of the system

- The inheritance mechanism allows compact codes and the overriding features allow localization of changes.

On the negative side are the following drawbacks of the OODBMS:

- Unlike the relational approach, which started out with a formal theory and a framework for a query language, there is no formal or accepted framework of OODBMS. This lack of a formal framework and query system means that the development of OODBMS will most likely be Darwinian, with the most popular becoming the de facto standard.
- Since each object is a self-contained unit, there is no means of showing relationships among a number of objects. Interobject reference is used to show such an association indirectly.
- Performance will likely be a problem. Techniques such as associative access and architecture features such as tagged architecture have to be investigated.
- In traditional database systems the user must know what the schema contains, such as names of relations and attributes, and pose queries and design programs using this knowledge. In an OODBMS the user must know what each object class is, as well as its methods, messages, and responses. This is not a light requirement[8] and may be the biggest stumbling block in the use of the object approach unless an intelligent user interface is provided with the database.

16.9 Summary

In this chapter we defined a knowledge base system as a computer system used for the management and manipulation of shared knowledge. We compared a knowledge base system with a DBMS and pointed out the similarities and differences. We considered the different schemes used to represent knowledge: the semantic network, first-order logic, rule-based system, frames, and procedural representation.

Expert systems are knowledge base systems wherein the knowledge of experts in a limited domain of application is stored; this knowledge can be used by appropriate inference procedures to solve problems in the domain. The knowledge in expert systems is usually stored as rules. The expert system also generates explanations, which can be employed to illustrate the rules used to answer a user query. Expert systems use forward chaining or backward chaining in their inference procedures.

SIMULA, a programming language for computer simulation, introduced the concept of object class. Class in SIMULA is an abstract data type mechanism and the object-oriented programming language is based on this concept (Gold 83). Objects can be considered uniform abstractions or representations of the storage and manipulation capabilities of a computing system. The set of operations performed by

[8] A case in point is the UNIX operating system. It started off with a lean and utilitarian system with very attractive features but it has become a dinosaur. The online help facility is of no use to a novice and the manuals are too large and badly organized.

They are looking at five major areas: programming languages; concurrency control; object-based management; software management; and user interface and environment.

16.8 Object Databases

In databases, we concern ourselves with the management and sharing of a large amount of reliable and persistent data. The relational system is suitable for an ad hoc query expressed in a query language such as SQL. However, such query languages are not suitable for application development. The application development language must be suitably integrated with the relational query language and should have a similar model of the computation being performed. Unfortunately whereas relational query languages are set oriented, application languages tend to be record oriented. Object orientation, with the ability to treat everything as objects, including programs and data, is therefore a promising avenue of research.

The object model and the object approacn have not been defined formally; consequently a large number of systems can rightly claim to be using the object model. The justification of the use of this approach in programming language is to provide an increased degree of abstraction. In the area of OS there is a constant need to reduce complexity in allowing concurrent tasks to share resources in an orderly manner and to communicate with each other. In the DBMS, there is a need to model complex entities such as CAD/CAM design data, office documents, and coauthored articles.

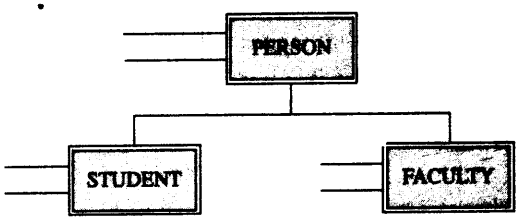
Along with the lack of a clear, well-defined and accepted object model, there is a lack of uniformity in the concept of an object-oriented database system. In a DBMS, the relationship between two record types may be statically established or based on the content. Relationships exist between classes due to the hierarchical structure and inheritance between subclass and superclass. In the object model, relationships may exist at object level via objects that know about each other and communicate via messages. However, content relationships between objects may not be allowed if the object paradigm is to be preserved. In a database system, all record instances share the same set of operations, which are implemented in the DBMS. In the object model, each object has its own set of operations and can be tailored to the object. However, to achieve efficiency, we use multiple inheritance, which creates its own set of problems. Database record instances are accessed based on the contents. In the object model, the contents of the object are encapsulated and not accessible; therefore, the identifiers are the only means of externally identifying an object instance.

Research projects in object databases can be classed as either an extension to existing systems or as an object-oriented DBMS (OODBMS). In the latter, the data model supports the object approach.

Extension to Existing Systems

POSTGRES (Ston 86a) (Ston 86b), designed by Stonebraker and his colleagues, extends the relational model by supporting abstract data types and procedures. The latter can be used to simulate objects.

Figure 16.14 Objects in the university database.



Class inheritance provides a method whereby a new class can be defined as a subclass of an existing class. It inherits not only the operations of the parent class but also its data structure (instance variable). It could be possible to add further data structures and operations on these structures in the subclass.

In **multiple inheritance** a subclass is considered to have not one parent class but multiple parent classes (Figure 16.15). Many of the OOP languages provide only single (or simple) inheritance. In the case of multiple inheritance, there would be the need to override one or more inherited methods and a method of resolving conflict in names of operations or instance variables. Conflict resolution would be by explicit disambuguation, default rules, or prefixing the name with that of the parent class.

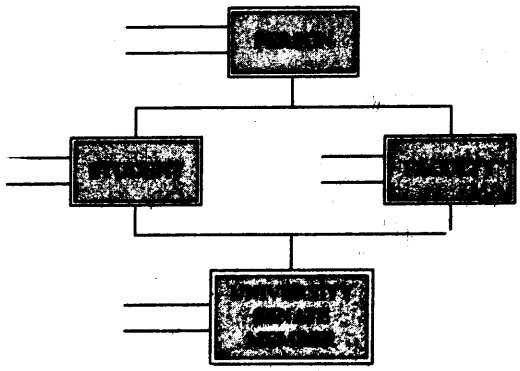
If a class has to be modified in the presence of existing instances of objects of the class and its subclasses, there is a need for some form of **object independence.**

In **partial inheritance** the subclass inherits only a subset of the data structures and operations from the parent class and suppresses the remaining.

Class inheritance is a static mechanism. In **dynamic inheritance,** an object changes its response to a message when it accepts new parts from other objects or when it changes its environment. The latter concept is similar to a programming language where the environment can be changed dynamically, as in PL/1. Similarly, a given text changes its fonts when a new style sheet is attached to the document.

The direction of research and the very concept of an object depend on the roots of the researcher. Researchers are discovering new ways to use the old concepts.

Figure 16.15 Example of multiple inheritance.



totally annihilated and no memory of such an object remains in some other object. If such a memory remains in the system, we have a problem of dangling pointers, which should not be allowed. The identifier of an object that ceases to exist may be reassigned depending on the implementation.

Object identifiers are useful for implementation and allow users to perform tests on the identity of an object. Nevertheless, they should not play a role in the model.

16.7.5 Object Class and Instantiation

As in traditional programming language, the notion of type is used to describe an object. It consists of two parts: the data and operation parts and their implementations, and the interface to the object that is visible from the outside. The data and the implementation of the operations on this data are private to the object. The operations that are implemented cater to the specified interface of the object.

Traditional programming language provides a number of data types such as integers, character strings, bit strings, floating point numbers, and so forth. These can be used as required by associating a name with an instance of this type. The instantiation can be static at compile time or dynamic at run time depending on the features provided by the language.

Similarly, in OOP, objects may be instantiated either statically at compile time or dynamically at run time. There could exist more than one object that recognizes and responds similarly to the same set of messages. These objects of the same object types are grouped together into a class of objects or simply as a class. Such objects have the same type of private memory, which is referred to by their methods using the same set of names. Each class has a name and is itself considered as an object belonging to a special system-defined class.

The collection of a group of identical objects into a class allows the sharing of common methods. The concept class thus groups together a set of externally visible operations, a set of corresponding hidden methods, and a set of private variables belonging to instances of the objects of the class. A new instance of an object in a class has its own private memory and shares the operations and the methods of the class.

16.7.6 Inheritance

In OOP, inheritance is used to allow different objects to share attributes and methods. One advantage of inheritance is lower development time due to program reusability.

In our university database (see Figure 16.14), The objects FACULTY and STUDENT are both specializations of the object PERSON and share some common traits. They both have a *Birthdate*, an *Address*, a *Home_Phone_Number*, *Next_of_Kin*, and so on. A number of operations could be performed on these items. For instance, one of these items could be updated. The program to implement these operations could be shared. Similarly, each of the objects STUDENT and FACULTY has certain special attributes, i.e., *Set_of_Grades* for STUDENT and *Salary* for FACULTY.

16.7.3 Database and Identity

Databases emerged to resolve the storage problem and facilitate the sharing of persistent objects. This required the support of the identity of an object not only in terms of its representation but also over time.

Every object is unique. However, we cannot, for example, distinguish between two 2d nails, nor do we bother to try. What is important for most applications is that they are 2d nails as opposed to 3d nails. In modeling a definite object for a particular application, we do not model all of its characteristics but only a subset of interest to the application. This subset may not be sufficient to bring out the uniqueness of the object. (For example, the 2d nails could have some characteristics that may identify one nail uniquely from another.) We also use some means to characterize abstract objects. It may also happen that the uniqueness of the object can only be established as a result of the object's relationship with another object.

Database systems use the concept of key attributes to distinguish individual records or tuples (persistent objects). The data values of the key attributes are thus mixed with the identity of the objects. This dictates that the value of the key attributes cannot be modified, even though they are descriptive data or artificially introduced data. The name of a department, for instance, is used as a key of the department and also used as a foreign key in the employee relation (object) to establish the relationship that an employee is assigned to a given department. Suppose the name of a department changes as a result of reorganization or modernization, say, from Quantitative Methods to Decision Science or from Personnel to Human Resources. This causes the problem of updating in the department object and all others referring to that object.

A change could be required in the choice of an identifier. Such a situation occurs when preexisting databases having similar classes of objects with different identifiers must be integrated. Two different divisions of a company, for instance, may use different identifiers for identifying employees. One division may use a locally generated sequential employee number; the other may choose the Social Security number. Another problem with this approach is that the individual attributes or any subset of attributes of a relation lack an identity.

In the object-oriented approach, a separate consistent mechanism is used to identify an object regardless of the actual method used in modeling the object or the attributes associated with the object (i.e., the descriptive data). An object system can then be defined to be made up of objects. In a consistent object system no two distinct objects have the same object identifiers, and for each existing object identifier there is a corresponding object. Two objects, O_1 and O_2, are identical if the identifiers for the objects are identical.

16.7.4 Implementation of Object Identifiers

The **object identifier** is best implemented using a system-generated surrogate. Such object identifiers, provided operations on them are allowed, need not be accessible to a user. The question as to what to do with an object identifier when the corresponding object ceases to exist is simply answered if the object is considered to be

same thing. Everything is identical with itself and with nothing else. But despite its simplicity, identity invites confusion. E.g. it may be asked: Of what use is the notion of identity if identifying an object with itself is trivial and identifying with anything else is false?[7]

Having defined objects still leaves open the question of the ability to distinguish objects from each other. This ability must be distinct from the state of the object or its location and at the same time allow different objects to be shared. When talking of identifying something, we do not necessarily mean locating a name, but the object associated with the name.

Addressability is a scheme of locating an object or providing access to an object and is dependent on the environment. Consider Professor Smith. Her students address her in a way that is different from the way her children do, which in turn is different from the way her friends and acquaintances address her. However, Professor Smith is the same person and she knows it!

Addressability is external to an object. Consider the method used in FORTRAN to access a file. A file number is mapped into a logical file name, which is mapped into a physical file name and a physical file. The file has an identity of its own, which is compared to the physical file name to ensure that the correct file is accessed. This highlights the concept that **identity** is internal to the object.

Programming languages use variable names to distinguish objects, which last for the duration of the execution of the program or portion thereof. Such variable names are defined by users to represent the identity of an object. An object required by more than one program module is made global among these modules. The binding of an object, which in this case is a storage location in real or virtual memory, is done either at compilation time or run time. If the same program is rerun the same variable is used over again for the same purpose; this fact keeps us from realizing that these objects are only temporary. The addressing of an object is thus merged with the identity of the corresponding object. Objects that persist over different executions of a program or that are passed from one program to another use a file system.

The use of variable names without some built-in representation of identity and operator to test and manipulate this abstraction can cause problems. This is the case when the same transient objects are referred to by different variable names and accessed in different ways. The concept of COMMON in FORTRAN is used to share objects among different program units and could refer to the same object, such as storage location in real or virtual memory, using different variable names with possibly different data types. This creates a great number of errors that are hard to detect. The concept of EQUIVALENCE allows objects to be shared among variables of the same program module. Without a test for establishing the identity of the object, a problem is created. Pascal addressed this problem by introducing the variant record type. Smalltalk provides a simple identity test expression of the form X == Y, where two variables, X and Y, are tested to determine if their identity is the same

[7] W. V. Quine, *Methods of Logic*, 4th ed. Cambridge: Harvard University Press, 1982.