Source: STANDARD HANDBOOK OF ELECTRONIC ENGINEERING

CHAPTER 18.5
DATABASE TECHNOLOGY

DATABASE OVERVIEW

Around 1964 a new term appeared in the computer literature to denote a new concept. The term was "database," and it was coined by workers in military information systems to denote collections of data shared by end users of time-sharing computer systems. The commercial data-processing world at the time was in the throes of "integrated data processing," and quickly appropriated "database" to denote the data collection that results from consolidating the data requirements of individual applications. Since that time, the term and the concept have become firmly entrenched in the computer world. Today, computer applications in which users access a database are called *database applications.* The *database management system*, or DBMS, has evolved to facilitate the development of database applications. The development of DBMS, in turn, has given rise to new languages, algorithms, and software techniques, which together make up what might be called a *database technology.* An overview of a typical DBMS is shown in Fig. 18.5.1.

Traditional data-processing applications used *master files* to maintain continuity between program runs. Master files "belonged to" applications, and the master files within an enterprise were often designed and maintained independently of one another. As a result, common data items often appeared in different master files, and the values of such items often did not agree. There was thus a requirement to consolidate the various master files into a single database, which could be centrally maintained and shared among various applications. Data consolidation was also required for the development of certain types of "management information" applications that were not feasible with fragmented master files. There was a requirement to raise the level of languages used to specify application procedures, and also to provide software for automatically transforming high-level specifications into equivalent low-level specifications. In the database context, this property of languages has come to be known as *data independence.* The consolidation of master files into databases had the undesirable side effect of increasing the potential for data loss and unauthorized data use. The requirement for data consolidation thus carried with it a requirement for tools and techniques to control the use of databases and to protect against their loss.

A DBMS is characterized by its *data-structure class*, that is, the class of data structures that it makes available to users for the formulation of applications. Most DBMSs distinguish between structure *instances* and structure *types*, the latter being abstractions of sets of structure instances. A DBMS also provides an implementation of its data-structure class, which is conceptually a mapping of the structures of the class into the structures of a lower-level class. The structures of the former class are often referred to as *logical* structures, whereas those of the latter are called *physical* structures. The data-structure classes are early systems, were derived from punched-card technology, and thus tended to be quite simple. A typical class was composed of *files* of *records* of a single type, with the record type being defined by an ordered set of fixed-length fields. Because of their regularity, such files are now referred to as *flat files.*

Database technology has produced a variety of improved data-structuring methods, many of which have been embodied in DBMS. Although many specific data-structure classes have been produced (essentially one class per system), these classes have tended to cluster into a small number of families, the most important of which are the *hierarchic*, the *network*, the *relational*, and the *semantic* families. These families have evolved more or less in the

Downloaded from Digital Engineering Library @ McGraw-Hill (www.digitalengineeringlibrary.com)
Copyright © 2004 The McGraw-Hill Companies. All rights reserved.
Any use is subject to the Terms of Use as given at the website.

DATABASE TECHNOLOGY

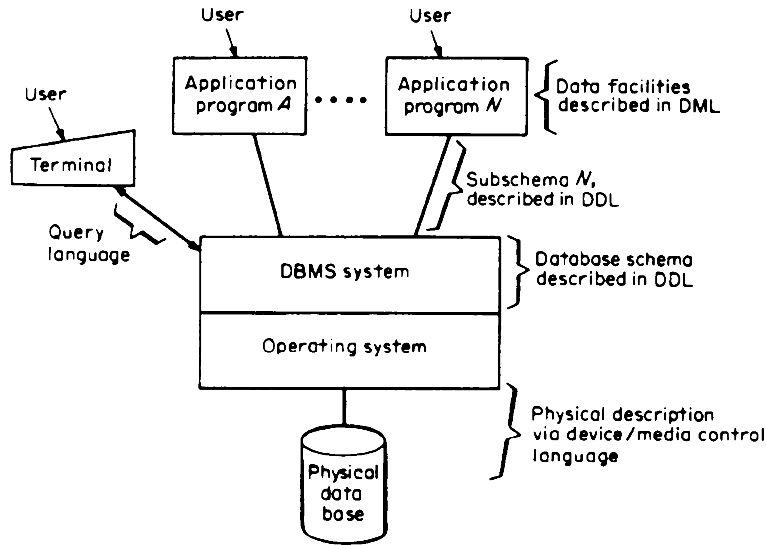

FIGURE 18.5.1 Overview of a typical database management system (DBMS).

order indicated, and all are represented in the data-structure classes of present-day DBMS. The use of large databases in distributed computing systems is sometimes called *Data Repository* or *Data Warehouse.*

HIERARCHIC DATA STRUCTURES

The *hierarchic data-structuring methods* that began to appear in the early 1960s provided some relief for the entity association problem. These methods were developed primarily to accommodate the variability that frequently occurs in the records of a file. For example, in the popular two-level hierarchic method, a record was divided into a *header* segment and a variable number of *trailer* segments of one or more types. The header segment represented attributes common to all entities of a set, while the trailer segments were used for the variably occurring attributes. The method was also capable of representing one-many associations between two sets of entities, by representing one set as header segments and the other as trailers and thus provided a primitive tool for data consolidation.

This two-level approach was expanded to *n*-level structures. These structures have also been implemented extensively on direct-access storage devices (DASD), which afford numerous additional representation possibilities. IMS was one of the first commercial systems to offer hierarchic data structuring and is often cited to illustrate the hierarchic structuring concept. The IMS equivalent of a file is the *physical database*, which consists of a set of hierarchical structured records of a single type. A record type is composed according to the following rules. The record type has a single type of *root* segment. The root segment type may have any number of *child* segment types. Each child of the root may also have any number of child segment types, and so on.

NETWORK DATA STRUCTURES

The first network structuring method to be developed for commercial data processing had its origins in the bill-of-materials application, which requires the representation of many-many associations between a set of parts and itself; e.g., a given part may simultaneously act as an assembly of other parts and as a component of other parts.

Downloaded from Digital Engineering Library @ McGraw-Hill (www.digitalengineeringlibrary.com)
Copyright © 2004 The McGraw-Hill Companies. All rights reserved.
Any use is subject to the Terms of Use as given at the website.

DATABASE TECHNOLOGY

This concept was used as the basis of a database language developed by the database task group (DBTG) of CODASYL in the late 1960s and early 1970s. This language introduced some new terminology and generalized some features, and has been implemented in a number of DBMSs.

RELATIONAL DATA STRUCTURES

In the mid-1960s, a number of investigators began to grow dissatisfied with the hardware orientation of then extent data-structuring methods and, in particular, with the manner in which pointers and similar devices for implementing entity associations were being exposed to the users. These investigators sought a way of raising the perceived level of data structures and at the same time bringing them closer to the way in which people look at information. Their efforts resulted in an *entity-set*-structuring method, wherein information is represented in a set of tables, with each table corresponding to a set of entities of a single type. The rows of a table correspond to the entities in the set, and the columns correspond to the attributes that characterize the entity set type.

Tables can be used to represent associations among entities. In this case, each row corresponds to an association, and the columns correspond to entity identifiers, i.e., entity attributes that can be used to uniquely identify entities. Additional columns may be used to record attributes of the association itself (as opposed to attributes of the associated entities). The key new concepts in the entity set methods were the simplicity of the structures it provided and the use of entity identifiers (rather than pointers or hardware-dictated structures) for representing entity associations. These concepts represented a major step forward in meeting the general goal of *data independence.*

Codd (1971) noted that an entity set could be viewed as a mathematical relation on a set of domains, where each domain corresponds to a different property of the entity set. Associations among entities could be similarly represented, with the domains in this case corresponding to entity identifiers. Codd defined a (data) *relation* to be a time-varying subset of the cartesian product of the members of the set of domain *n*-tuples (or simply *tuples*). Codd proposed that relations be built exclusively on domains of elementary values—integers, character strings, and so forth. He called such relations *normalized relations* and the process of converting relations to normalized form, *normalization.* Virtually all work done with relations has been with normalized relations.

Codd postulated levels of normalization called *normal forms.* An unconstrained normalized relation is in *first normal form* (1NF). A relation in 1NF in which all nonkey domains are functionally dependent on (i.e., have their values determined by) the entire key are in *second normal form* (2NF), which solves the problem of parasitic entity representation. A relation in 2NF in which all nonkey domains are dependent *only* on the key is the *third normal form* (3NF), which solves the problem of masquerading entities. As part of the development of the relational method, Codd postulated a *relational algebra*, i.e., a set of operations on relations that is closed in the sense of a traditional algebra, and thereby provided an important formal vehicle for carrying out a variety of research in data structures and systems. In addition to the conventional set operations, the relational algebra provides such operations as *restriction*, to delete selected tuples of a relation; *projection*, to delete selected domains of a relation; and *join*, to join two relations into one. Codd also proposed a *relational calculus,* whose distinguishing feature is the method used to designate sets of tuples. The method is patterned after the predicate calculus and makes use of free and bound variables and the universal and existential quantifiers.

Codd characterized his methodology as a *data model*, and thereby provided a concise term for an important but previously unarticulated database concept, namely, the *combination* of a class of data structures and the operation allowed on the structures of the class. (A similar concept, the *abstract data type* or *data abstraction*, has evolved elsewhere in software technology.) The term "model" has been applied retroactively to early data-structuring methods, so that, for example, we now speak of hierarchic models and network models, as well as the relational model. The term is now generally used to denote an abstract data-structure class, although there is a growing realization that it should embrace operations as well as structures. The relational model has been implemented in a number of DBMSs.

SEMANTIC DATA STRUCTURES

During the evolution of the hierarchic, network, and relational methods, it gradually became apparent that building a database was in fact equivalent to building a model of an enterprise and that databases could be developed more or less independently of applications simply by studying the enterprise. The notion of a *conceptual schema*

Downloaded from Digital Engineering Library @ McGraw-Hill (www.digitalengineeringlibrary.com)
Copyright © 2004 The McGraw-Hill Companies. All rights reserved.
Any use is subject to the Terms of Use as given at the website.

for the application-independent modeling of an enterprise and various *external schemata* derivable from the conceptual schema for expressing data requirements of specific applications is the outgrowth of this view.

Application-independent modeling has produced a spate of *semantic* data models and debate over which of these is best for modeling "reality." One of the most successful semantic models is the *entity-relationship model*, which provides data constructs at two levels: the *conceptual* level, whose constructs include entities, relationships (*n*-ary associations among entities), value sets, and attributes; and the *representation* level, in which conceptual constructs are mapped into tables.

DATA DEFINITION AND DATA-DEFINITION LANGUAGES

The history of computer applications has been marked by a steady increase in the level of the language used to implement applications. In database technology, this trend is manifested in the development of high-level data-definition languages and data-manipulation languages. A *data-definition language* (DDL) provides the DBMS user with a way to declare the attributes of structure types within the database, and thus enable the system to perform implicitly many operations (e.g., name resolution, data-type checking) that would otherwise have to be invoked explicitly. A DDL typically provides for the definition of both logical and physical data attributes as well as for the definition of different *views* of the (logical) data. The latter are useful in limiting or tailoring the way in which specific programs or end users look at the database. A *data-manipulation language* (DML) provides the user with a way to express operations on the data structure instances of a database, using names previously established through data definitions. Data-manipulation facilities are of two general types: host-language and self-contained.

A *host-language* facility permits the manipulation of databases through programs written in conventional procedural languages such as COBOL or PL/I. It provides statements that the user may embed in a program at the points were database operations are to be performed. When such a statement is encountered control is transferred to the database system, which performs the operation and returns the results (data and return codes) to the program in prearranged main-storage locations.

A *self-contained* facility permits the manipulation of the database through a high-level, nonprocedural language, which is independent of any procedural language, i.e., whose language is self-contained. An important type of self-contained facility is the *query facility*, which enables "casual" users to access a database without the mediation of a professional programmer. Other types of self-contained facility are available for performing generalizable operations on data, such as sorting report generation, and data translation.

REPORT PROGRAM GENERATORS

The use of fixed files for reporting is found in the *report program generator*, a software package intended primarily for the production of reports from formatted files. Attributes of the source files and the desired reports are described by the user in a simple declarative language, and this description is then processed by a compiler to generate a program that, when run, produces the desired reports. A key concept of the report program generator is the use of a fixed structure for the generated program, consisting of input, calculation, and output phases. Such a structure limits the transformations that can be carried out with a single generated program, but it has nevertheless proved to be remarkably versatile. (Report program generators are routinely used for file maintenance as well as for report generation.) The fixed structure of the generated program imposes a discipline on the user, which enables the user to produce a running program much more quickly than could otherwise be done with conventional languages. Report program generators were especially popular in smaller installations where conventional programming talent is scarce, and in some installations it was the only "programming language" used.

The report program generators and the formatted file systems were the precursors of the contemporary DBMS query facility. A query processor is in effect a generalized routine that is particularized to a specific application (i.e., the user's query) by the parameters (data names, Boolean predicates, and so forth) appearing in the query. Query facilities are more advanced than most early generalized routines in that they provide online (as opposed to batch) access to databases (as opposed to individual files). The basic concept is unchanged, however, and the lessons learned in implementing the generalized routines, and especially in reconciling ease of use with acceptable performance, have been directly applicable to query-language processors.

Downloaded from Digital Engineering Library @ McGraw-Hill (www.digitalengineeringlibrary.com)
Copyright © 2004 The McGraw-Hill Companies. All rights reserved.
Any use is subject to the Terms of Use as given at the website.

DATABASE TECHNOLOGY

PROGRAM ISOLATION

Most DBMSs permit a database to be accessed concurrently by a number of users. If this access is not controlled, the consistency of the data can be compromised (e.g., lost updates) or the logic of programs can be affected (e.g., nonrepeatable read operations). With program isolation, records are locked for a program upon updating any item within the record and unlocked when the program reaches a *synchpoint*, i.e., a point at which the changes made by the program are committed to the database. Deadlocks can occur and are resolved by selecting one of the deadlocked programs and restarting it at its most recent synchpoint.

AUTHORIZATION

Consolidated data often constitute sensitive information, which the user may not want divulged to other than authorized people, for reasons of national security, competitive advantage, or personal privacy. DBMSs, therefore, provide mechanisms for limiting data access to properly authorized persons. A system administrator can grant specific capabilities with respect to specific data objects to specific users. Grantable capabilities with respect to relations include the capability to read from the relation, to insert tuples, to delete tuples, to update specific fields, and to delete the relation. The holder of a capability may also be given authority to grant that capability to others, so that authorization tasks may be delegated to different individuals within an organization.

Downloaded from Digital Engineering Library @ McGraw-Hill (www.digitalengineeringlibrary.com)
Copyright © 2004 The McGraw-Hill Companies. All rights reserved.
Any use is subject to the Terms of Use as given at the website.