DATABASE

1.1 DEFINITION A **database** is an organized collection of <u>data</u>. The data are typically organized to model aspects of reality in a way that supports processes requiring information. For example, modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

Database management systems (**DBMS**s) are specially designed software applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is a <u>software</u> system designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include <u>MySQL</u>, <u>PostgreSQL</u>, <u>Microsoft SQL Server</u>, <u>Oracle, SAP</u> and <u>IBM DB2</u>. A database is not generally <u>portable</u> across different DBMSs, but different DBMSs can interoperate by using <u>standards</u> such as <u>SQL</u> and <u>ODBC</u> or <u>JDBC</u> to allow a single application to work with more than one DBMS. Database management systems are often classified according to the <u>database model</u> that they support; the most popular database systems since the 1980s have all supported the <u>relational model</u> as represented by the <u>SQL</u> language.

1.2 Terminology and overview

Formally, "database" refers to the data themselves and supporting data structures. Databases are created to operate large quantities of information by inputting, storing, retrieving and managing that information. Databases are set up so that one set of software programs provides all users with access to all the data.

A "database management system" is a suite of computer software providing the interface between <u>users</u> and a database or databases. Because they are so closely related, the term "database" when used casually often refers to both a DBMS and the data it manipulates.

Outside the world of professional <u>information technology</u>, the term *database* is sometimes used casually to refer to any collection of data (perhaps a <u>spreadsheet</u>, maybe even a card index). This article is concerned only with databases where the size and usage requirements necessitate use of a database management system.^[11]

The interactions catered for by most existing DBMSs fall into four main groups:

- **Data definition** Defining new data structures for a database, removing data structures from the database, modifying the structure of existing data.
- Update Inserting, modifying, and deleting data.

- **Retrieval** Obtaining information either for end-user queries and reports or for processing by applications.
- Administration Registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control, and recovering information if the system fails.

A DBMS is responsible for maintaining the integrity and security of stored data, and for recovering information if the system fails.

Both a database and its DBMS conform to the principles of a particular <u>database</u> <u>model</u>.^[2] "Database system" refers collectively to the database model, database management system, and database.^[3]

Physically, database <u>servers</u> are dedicated computers that hold the actual databases and run only the DBMS and related software. Database servers are usually <u>multiprocessor</u> computers, with generous memory and <u>RAID</u> disk arrays used for stable storage. RAID is used for recovery of data if any of the disks fail. Hardware database accelerators, connected to one or more servers via a high-speed channel, are also used in large volume transaction processing environments. DBMSs are found at the heart of most <u>database applications</u>. DBMSs may be built around a custom <u>multitasking kernel</u> with built-in <u>networking</u> support, but modern DBMSs typically rely on a standard <u>operating system</u> to provide these functions. Since DBMSs comprise a significant <u>economical market</u>, computer and storage vendors often take into account DBMS requirements in their own development plans.¹

Databases and DBMSs can be categorized according to the database model(s) that they support (such as relational or XML), the type(s) of computer they run on (from a server cluster to a mobile phone), the <u>query language(s)</u> used to access the database (such as SQL or <u>XQuery</u>), and their internal engineering, which affects performance, <u>scalability</u>, resilience, and security.

Applications

Databases are used to support internal operations of organizations and to underpin online interactions with customers and suppliers (see <u>Enterprise software</u>).

Databases are used to hold administrative information and more specialized data, such as engineering data or economic models. Examples of database applications include computerized <u>library</u> systems, <u>flight reservation systems</u> and computerized <u>parts inventory systems</u>.

General-purpose and special-purpose DBMSs]

A DBMS has evolved into a complex software system and its development typically requires thousands of person-years of development effort.^[4] Some general-purpose DBMSs such as <u>Adabas</u>, <u>Oracle</u> and DB2 have been undergoing upgrades since the 1970s. General-purpose DBMSs aim to meet the needs of as many applications as possible, which adds to the complexity. However, the fact that their development cost can be spread over a large number of users means that they are often the most cost-effective approach. However, a general-purpose DBMS is not always the optimal solution: in some cases a general-purpose DBMS may introduce unnecessary overhead. Therefore, there are many examples of systems that use special-purpose databases. A common example is an <u>email</u> system: email systems are designed to optimize the handling of email messages, and do not need significant portions of a general-purpose DBMS functionality.

Many databases have <u>application software</u> that accesses the database on behalf of end-users, without exposing the DBMS interface directly. Application programmers may use a <u>wire protocol</u> directly, or more likely through an <u>application programming interface</u>. Database designers and database administrators interact with the DBMS through dedicated interfaces to build and maintain the applications' databases, and thus need some more knowledge and understanding about how DBMSs operate and the DBMSs' external interfaces and tuning parameters.

History

Following the <u>technology</u> progress in the areas of <u>processors</u>, <u>computer memory</u>, <u>computer storage</u> and <u>computer networks</u>, the sizes, capabilities, and performance of databases and their respective DBMSs have grown in orders of magnitude. The development of database technology can be divided into three eras based on data model or structure: <u>navigational</u>, SQL/relational, and post-relational.

The two main early navigational data models were the <u>hierarchical model</u>, epitomized by IBM's IMS system, and the <u>CODASYL</u> model (<u>network model</u>), implemented in a number of products such as <u>IDMS</u>.

The <u>relational model</u>, first proposed in 1970 by <u>Edgar F. Codd</u>, departed from this tradition by insisting that applications should search for data by content, rather than by following links. The relational model employs sets of ledger-style tables, each used for a different type of entity. Only in the mid-1980s did computing hardware become powerful enough to allow the wide deployment of relational systems

(DBMSs plus applications). By the early 1990s, however, relational systems dominated in all large-scale data processing applications, and as of 2014 they remain dominant except in niche areas. The dominant database language, standardised SQL for the relational model, has influenced database languages for other data models.

<u>Object databases</u> were developed in the 1980s to overcome the inconvenience of <u>object-relational impedance mismatch</u>, which led to the coining of the term "post-relational" and also the development of hybrid <u>object-relational databases</u>.

The next generation of post-relational databases in the late 2000s became known as <u>NoSQL</u> databases, introducing fast <u>key-value stores</u> and <u>document-oriented</u> <u>databases</u>. A competing "next generation" known as <u>NewSQL</u> databases attempted new implementations that retained the relational/SQL model while aiming to match the high performance of NoSQL compared to commercially available relational DBMSs.

1960s, navigational DBMS

The introduction of the term *database* coincided with the availability of directaccess storage (disks and drums) from the mid-1960s onwards. The term represented a contrast with the tape-based systems of the past, allowing shared interactive use rather than daily <u>batch processing</u>. The <u>Oxford English dictionary</u> cites^[6] a 1962 report by the System Development Corporation of California as the first to use the term "data-base" in a specific technical sense.

As computers grew in speed and capability, a number of general-purpose database systems emerged; by the mid-1960s a number of such systems had come into commercial use. Interest in a standard began to grow, and <u>Charles Bachman</u>, author of one such product, the <u>Integrated Data Store</u> (IDS), founded the "Database Task Group" within <u>CODASYL</u>, the group responsible for the creation and standardization of <u>COBOL</u>. In 1971 the Database Task Group delivered their standard, which generally became known as the "CODASYL approach", and soon a number of commercial products based on this approach entered the market.

The CODASYL approach relied on the "manual" navigation of a linked data set which was formed into a large network. Applications could find records by one of three methods:

1. Use of a primary key (known as a CALC key, typically implemented by hashing)

- 2. Navigating relationships (called sets) from one record to another
- 3. Scanning all the records in a sequential order

Later systems added B-Trees to provide alternate access paths. Many CODASYL databases also added a very straightforward query language. However, in the final tally, CODASYL was very complex and required significant training and effort to produce useful applications.

IBM also had their own DBMS in 1968, known as Information Management System (IMS). IMS was a development of software written for the Apollo program on the System/360. IMS was generally similar in concept to CODASYL, but used a strict hierarchy for its model of data navigation instead of CODASYL's network model. Both concepts later became known as navigational databases due to the way data was accessed, and Bachman's 1973 <u>Turing Award</u> presentation was *The Programmer as Navigator*. IMS is classified as a <u>hierarchical database</u>. IDMS and <u>Cincom Systems' TOTAL</u> database are classified as network databases. IMS remains in use as of 2014.

1970s, relational DBMS

<u>Edgar Codd</u> worked at IBM in <u>San Jose, California</u>, in one of their offshoot offices that was primarily involved in the development of <u>hard disk</u> systems. He was unhappy with the navigational model of the CODASYL approach, notably the lack of a "search" facility. In 1970, he wrote a number of papers that outlined a new approach to database construction that eventually culminated in the groundbreaking *A Relational Model of Data for Large Shared Data Banks*.^[8]

In this paper, he described a new system for storing and working with large databases. Instead of records being stored in some sort of <u>linked list</u> of free-form records as in CODASYL, Codd's idea was to use a "<u>table</u>" of fixed-length records, with each table used for a different type of entity. A linked-list system would be very inefficient when storing "sparse" databases where some of the data for any one record could be left empty. The relational model solved this by splitting the data into a series of normalized tables (or *relations*), with optional elements being moved out of the main table to where they would take up room only if needed. Data may be freely inserted, deleted and edited in these tables, with the DBMS doing whatever maintenance needed to present a table view to the application/user.

In the <u>relational model</u>, related records are linked together with a "key". The relational model also allowed the content of the database to evolve without

constant rewriting of links and pointers. The relational part comes from entities referencing other entities in what is known as one-to-many relationship, like a traditional hierarchical model, and many-to-many relationship, like a navigational (network) model. Thus, a relational model can express both hierarchical and navigational models, as well as its native tabular model, allowing for pure or combined modeling in terms of these three models, as the application requires.

For instance, a common use of a database system is to track information about users, their name, login information, various addresses and phone numbers. In the navigational approach all of these data would be placed in a single record, and unused items would simply not be placed in the database. In the relational approach, the data would be *normalized* into a user table, an address table and a phone number table (for instance). Records would be created in these optional tables only if the address or phone numbers were actually provided.

Linking the information back together is the key to this system. In the relational model, some bit of information was used as a "key", uniquely defining a particular record. When information was being collected about a user, information stored in the optional tables would be found by searching for this key. For instance, if the login name of a user is unique, addresses and phone numbers for that user would be recorded with the login name as its key. This simple "re-linking" of related data back into a single collection is something that traditional computer languages are not designed for.

Just as the navigational approach would require programs to loop in order to collect records, the relational approach would require loops to collect information about any *one* record. Codd's solution to the necessary looping was a set-oriented language, a suggestion that would later spawn the ubiquitous SQL. Using a branch of mathematics known as <u>tuple calculus</u>, he demonstrated that such a system could support all the operations of normal databases (inserting, updating etc.) as well as providing a simple system for finding and returning *sets* of data in a single operation.

Codd's paper was picked up by two people at Berkeley, <u>Eugene Wong</u> and <u>Michael</u> <u>Stonebraker</u>. They started a project known as <u>INGRES</u> using funding that had already been allocated for a geographical database project and student programmers to produce code. Beginning in 1973, INGRES delivered its first test products which were generally ready for widespread use in 1979. INGRES was similar to <u>System R</u> in a number of ways, including the use of a "language" for <u>data access</u>, known as <u>QUEL</u>. Over time, INGRES moved to the emerging SQL standard.

IBM itself did one test implementation of the relational model, <u>PRTV</u>, and a production one, <u>Business System 12</u>, both now discontinued. <u>Honeywell</u> wrote <u>MRDS</u> for <u>Multics</u>, and now there are two new implementations: <u>Alphora</u> <u>Dataphor</u> and <u>Rel</u>. Most other DBMS implementations usually called *relational* are actually SQL DBMSs.

In 1970, the University of Michigan began development of the <u>MICRO</u> <u>Information Management System</u> based on D.L. Childs' Set-Theoretic Data model. Micro was used to manage very large data sets by the <u>US Department of Labor</u>, the <u>U.S. Environmental Protection Agency</u>, and researchers from the <u>University of</u> <u>Alberta</u>, the <u>University of Michigan</u>, and <u>Wayne State University</u>. It ran on IBM mainframe computers using the <u>Michigan Terminal System</u>. The system remained in production until 1998.