SUBJECT-DATABASE MANAGEMENT SYSTM

[BCA-II YR]

In computing, a **database** is an organized collection of data or a type of data store based on the use of a **database management system** (**DBMS**), the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a **database system**. Often the term "database" is also used loosely to refer to any of the DBMS, the database system or an application associated with the database.

Small databases can be stored on a file system, while large databases are hosted on computer clusters or cloud storage. The design of databases spans formal techniques and practical considerations, including data modeling, efficient data representation and storage, query languages, security and privacy of sensitive data, and distributed computing issues, including supporting concurrent access and fault tolerance.

Computer scientists may classify database management systems according to the database models that they support. Relational databases became dominant in the 1980s. These model data as rows and columns in a series of tables, and the vast majority use SQL for writing and querying data. In the 2000s, non-relational databases became popular, collectively referred to as NoSQL, because they use different query languages.

Terminology and overview

Formally, a "database" refers to a set of related data accessed through the use of a "database management system" (DBMS), which is an integrated set of computer software that allows users to interact with one or more databases and provides access to all of the data contained in the database (although restrictions may exist that limit access to particular data). The DBMS provides various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organized.

Because of the close relationship between them, the term "database" is often used casually to refer to both a database and the DBMS used to manipulate it.

Outside the world of professional information technology, the term *database* is often used to refer to any collection of related data (such as a spreadsheet or a card index) as size and usage requirements typically necessitate use of a database management system.[1]

Existing DBMSs provide various functions that allow management of a database and its data which can be classified into four main functional groups:

- **Data definition** – Creation, modification and removal of definitions that detail how the data is to be organized.
- **Update** – Insertion, modification, and deletion of the data itself.[2]

- **Retrieval** – Selecting data according to specified criteria (e.g., a query, a position in a hierarchy, or a position in relation to other data) and providing that data either directly to the user, or making it available for further processing by the database itself or by other applications. The retrieved data may be made available in a more or less direct form without modification, as it is stored in the database, or in a new form obtained by altering it or combining it with existing data from the database.[3]
- **Administration** – Registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control, and recovering information that has been corrupted by some event such as an unexpected system failure.[4]

Both a database and its DBMS conform to the principles of a particular database model.[5] "Database system" refers collectively to the database model, database management system, and database.[6]

Physically, database servers are dedicated computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with generous memory and RAID disk arrays used for stable storage. 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 database applications. DBMSs may be built around a custom multitasking kernel with built-in networking support, but modern DBMSs typically rely on a standard operating system to provide these functions[]

Since DBMSs comprise a significant market, computer and storage vendors often take into account DBMS requirements in their own development plans.[7]

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 query language(s) used to access the database (such as SQL or XQuery), and their internal engineering, which affects performance, scalability, resilience, and security.

History

The sizes, capabilities, and performance of databases and their respective DBMSs have grown in orders of magnitude. These performance increases were enabled by the technology progress in the areas of processors, computer memory, computer storage, and computer networks. The concept of a database was made possible by the emergence of direct access storage media such as magnetic disks, which became widely available in the mid-1960s; earlier systems relied on sequential storage of data on magnetic tape. The subsequent development of database technology can be divided into three eras based on data model or structure: navigational,[8] SQL/relational, and post-relational.

The two main early navigational data models were the hierarchical model and the CODASYL model (network model). These were characterized by the use

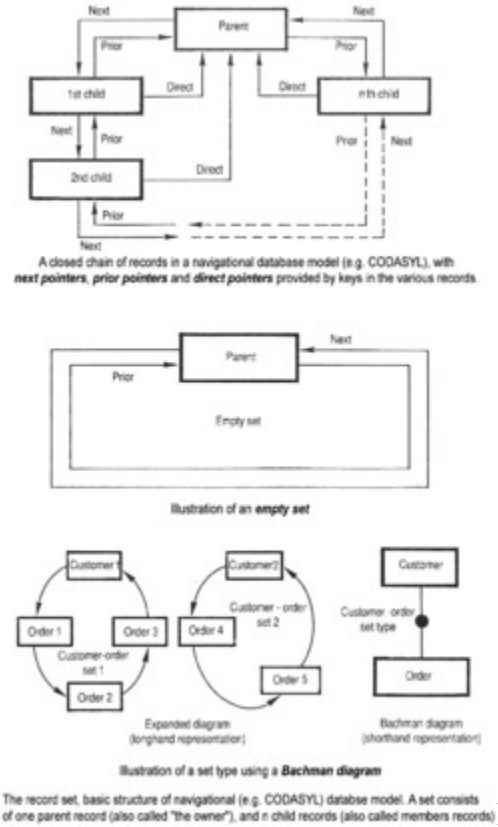
of pointers (often physical disk addresses) to follow relationships from one record to another.

The relational model, first proposed in 1970 by Edgar F. Codd, 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 2018 they remain dominant: IBM Db2, Oracle, MySQL, and Microsoft SQL Server are the most searched DBMS.[9] The dominant database language, standardized SQL for the relational model, has influenced database languages for other data models.[*citation needed*]

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

The next generation of post-relational databases in the late 2000s became known as NoSQL databases, introducing fast key–value stores and document-oriented databases. A competing "next generation" known as NewSQL 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



Basic structure of navigational CODASYL database model

The introduction of the term *database* coincided with the availability of direct-access 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 batch processing. The Oxford English Dictionary cites a 1962 report by the System Development Corporation of California as the first to use the term "data-base" in a specific technical sense.[10]

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 Charles Bachman, author of one such product, the Integrated Data Store (IDS), founded the Database Task Group within CODASYL, the group responsible for the creation and standardization of COBOL. 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 offered applications the ability to navigate around 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 declarative query language for end users (as distinct from the navigational API). However, CODASYL databases were complex and required significant training and effort to produce useful applications.

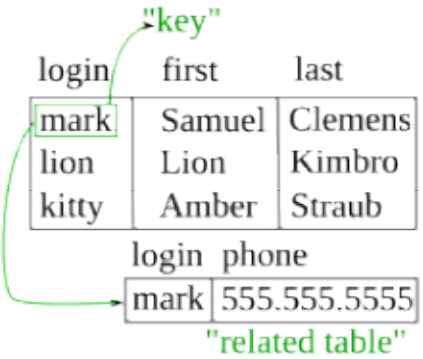
IBM also had its own DBMS in 1966, known as Information Management System (IMS). IMS was a development of software written for the Apollo program on the System/360. IMS was generally similar in concept to CODASYL, but used a strict hierarchy for its model of data navigation instead of CODASYL's network model. Both concepts later became known as navigational databases due to the way data was accessed: the term was popularized by Bachman's 1973 Turing Award presentation *The Programmer as Navigator*. IMS is classified by IBM as a hierarchical database. IDMS and Cincom Systems' TOTAL databases are classified as network databases. IMS remains in use as of 2014.[11]

1970s, relational DBMS

Edgar F. Codd worked at IBM in San Jose, California, in one of their offshoot offices that were primarily involved in the development of hard disk 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*.[12]

In this paper, he described a new system for storing and working with large databases. Instead of records being stored in some sort of linked list of free-form records as in CODASYL, Codd's idea was to organize the data as a number of "tables", each table being used for a different type of entity. Each table would contain a fixed number of columns containing the attributes of the entity. One or more columns of each table were designated as a primary key by which the rows of the table could be uniquely identified; cross-references between tables always used these primary keys, rather than disk addresses, and queries would join tables based on these key relationships, using a set of operations based on the mathematical system of relational calculus (from which the model takes its name). Splitting the data into a set of normalized tables (or *relations*) aimed to ensure that each "fact" was only stored once, thus simplifying update operations. Virtual tables called *views* could present the data in different ways for different users, but views could not be directly updated.

Codd used mathematical terms to define the model: relations, tuples, and domains rather than tables, rows, and columns. The terminology that is now familiar came from early implementations. Codd would later criticize the tendency for practical implementations to depart from the mathematical foundations on which the model was based.



In the relational model, records are "linked" using virtual keys not stored in the database but defined as needed between the data contained in the records.

The use of primary keys (user-oriented identifiers) to represent cross-table relationships, rather than disk addresses, had two primary motivations. From an engineering perspective, it enabled tables to be relocated and resized without expensive database reorganization. But Codd was more interested in the difference in semantics: the use of explicit identifiers made it easier to define update operations with clean mathematical definitions, and it also enabled query operations to be defined in terms of the established discipline of first-order predicate calculus; because these operations have clean mathematical properties, it becomes possible to rewrite queries in provably correct ways, which is the basis of query optimization. There is no loss of expressiveness compared with the hierarchic or network models, though the connections between tables are no longer so explicit.

In the hierarchic and network models, records were allowed to have a complex internal structure. For example, the salary history of an employee might be represented as a "repeating group" within the employee record. In the relational model, the process of normalization led to such internal structures being replaced by data held in multiple tables, connected only by logical keys.

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 this data would be placed in a single variable-length record. 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.

As well as identifying rows/records using logical identifiers rather than disk addresses, Codd changed the way in which applications assembled data from multiple records. Rather than requiring applications to gather data one record at a time by navigating the links, they would use a declarative query language that expressed what data was required, rather than the access path by which it should be found. Finding an efficient access path to the data became the responsibility of the database management system, rather than the application programmer. This process, called query optimization, depended on the fact that queries were expressed in terms of mathematical logic.

Codd's paper was picked up by two people at Berkeley, Eugene Wong and Michael Stonebraker. They started a project known as INGRES 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 System R in a number of ways, including the use of a "language" for data access, known as QUEL. Over time, INGRES moved to the emerging SQL standard.

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

In 1970, the University of Michigan began development of the MICRO Information Management System[13] based on D.L. Childs' Set-Theoretic Data model.[14][15][16] MICRO was used to manage very large data sets by the US Department of Labor, the U.S. Environmental Protection Agency, and researchers from the University of Alberta, the University of Michigan, and Wayne State University. It ran on IBM mainframe computers using the Michigan Terminal System.[17] The system remained in production until 1998.

Integrated approach

Main article: Database machine

In the 1970s and 1980s, attempts were made to build database systems with integrated hardware and software. The underlying philosophy was that such integration would provide higher performance at a lower cost. Examples were IBM System/38, the early offering of Teradata, and the Britton Lee, Inc. database machine.

Another approach to hardware support for database management was ICL's CAFS accelerator, a hardware disk controller with programmable search capabilities. In the long term, these efforts were generally unsuccessful because specialized database machines could not keep pace with the rapid development and progress of general-purpose computers. Thus most database systems nowadays are software systems running on general-purpose hardware, using general-purpose computer data storage. However, this idea is still pursued in certain applications by some companies like Netezza and Oracle (Exadata).

Late 1970s, SQL DBMS

IBM started working on a prototype system loosely based on Codd's concepts as *System R* in the early 1970s. The first version was ready in 1974/5, and work then started on multi-table systems in which the data could be split so that all of the data for a record (some of which is optional) did not have to be stored in a single large "chunk". Subsequent multi-user versions were tested by customers in 1978 and 1979, by which time a standardized query language – SQL[*citation needed*] – had been added. Codd's ideas were establishing themselves as both workable and superior to CODASYL, pushing

IBM to develop a true production version of System R, known as *SQL/DS*, and, later, *Database 2* (IBM Db2).

Larry Ellison's Oracle Database (or more simply, Oracle) started from a different chain, based on IBM's papers on System R. Though Oracle V1 implementations were completed in 1978, it was not until Oracle Version 2 when Ellison beat IBM to market in 1979.[18]

Stonebraker went on to apply the lessons from INGRES to develop a new database, Postgres, which is now known as PostgreSQL. PostgreSQL is often used for global mission-critical applications (the .org and .info domain name registries use it as their primary data store, as do many large companies and financial institutions).

In Sweden, Codd's paper was also read and Mimer SQL was developed in the mid-1970s at Uppsala University. In 1984, this project was consolidated into an independent enterprise.

Another data model, the entity–relationship model, emerged in 1976 and gained popularity for database design as it emphasized a more familiar description than the earlier relational model. Later on, entity–relationship constructs were retrofitted as a data modeling construct for the relational model, and the difference between the two has become irrelevant.[*citation needed*]

1980s, on the desktop

The 1980s ushered in the age of desktop computing. The new computers empowered their users with spreadsheets like Lotus 1-2-3 and database software like dBASE. The dBASE product was lightweight and easy for any computer user to understand out of the box. C. Wayne Ratliff, the creator of dBASE, stated: "dBASE was different from programs like BASIC, C, FORTRAN, and COBOL in that a lot of the dirty work had already been done. The data manipulation is done by dBASE instead of by the user, so the user can concentrate on what he is doing, rather than having to mess with the dirty details of opening, reading, and closing files, and managing space allocation."[19] dBASE was one of the top selling software titles in the 1980s and early 1990s.

1990s, object-oriented

The 1990s, along with a rise in object-oriented programming, saw a growth in how data in various databases were handled. Programmers and designers began to treat the data in their databases as objects. That is to say that if a person's data were in a database, that person's attributes, such as their address, phone number, and age, were now considered to belong to that person instead of being extraneous data. This allows for relations between data to be related to objects and their attributes and not to individual fields.[20] The term "object–relational impedance mismatch" described the inconvenience of translating between programmed objects and database tables. Object databases and object–relational databases attempt to solve this problem by providing an object-oriented language (sometimes as extensions to SQL) that programmers can

use as alternative to purely relational SQL. On the programming side, libraries known as object–relational mappings (ORMs) attempt to solve the same problem.

2000s, NoSQL and NewSQL

Main articles: NoSQL and NewSQL

XML databases are a type of structured document-oriented database that allows querying based on XML document attributes. XML databases are mostly used in applications where the data is conveniently viewed as a collection of documents, with a structure that can vary from the very flexible to the highly rigid: examples include scientific articles, patents, tax filings, and personnel records.

NoSQL databases are often very fast, do not require fixed table schemas, avoid join operations by storing denormalized data, and are designed to scale horizontally.

In recent years, there has been a strong demand for massively distributed databases with high partition tolerance, but according to the CAP theorem, it is impossible for a distributed system to simultaneously provide consistency, availability, and partition tolerance guarantees. A distributed system can satisfy any two of these guarantees at the same time, but not all three. For that reason, many NoSQL databases are using what is called eventual consistency to provide both availability and partition tolerance guarantees with a reduced level of data consistency.

NewSQL is a class of modern relational databases that aims to provide the same scalable performance of NoSQL systems for online transaction processing (read-write) workloads while still using SQL and maintaining the ACID guarantees of a traditional database system.

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

Databases are used to hold administrative information and more specialized data, such as engineering data or economic models. Examples include computerized library systems, flight reservation systems, computerized parts inventory systems, and many content management systems that store websites as collections of webpages in a database.

Classification

One way to classify databases involves the type of their contents, for example: bibliographic, document-text, statistical, or multimedia objects. Another way is by their application area, for example: accounting, music compositions, movies, banking, manufacturing, or insurance. A third way is by some technical aspect, such as the database structure or interface type. This section lists a few of the adjectives used to characterize different kinds of databases.

- An in-memory database is a database that primarily resides in main memory, but is typically backed-up by non-volatile computer data storage. Main memory databases are faster than disk databases, and so are often used where response time is critical, such as in telecommunications network equipment.
- An active database includes an event-driven architecture which can respond to conditions both inside and outside the database. Possible uses include security monitoring, alerting, statistics gathering and authorization. Many databases provide active database features in the form of database triggers.
- A cloud database relies on cloud technology. Both the database and most of its DBMS reside remotely, "in the cloud", while its applications are both developed by programmers and later maintained and used by end-users through a web browser and Open APIs.
- Data warehouses[*citation needed*] archive data from operational databases and often from external sources such as market research firms. The warehouse becomes the central source of data for use by managers and other end-users who may not have access to operational data. For example, sales data might be aggregated to weekly totals and converted from internal product codes to use UPCs so that they can be compared with ACNielsen data. Some basic and essential components of data warehousing include extracting, analyzing, and mining data, transforming, loading, and managing data so as to make them available for further use.
- A deductive database combines logic programming with a relational database.
- A distributed database is one in which both the data and the DBMS span multiple computers.
- A document-oriented database is designed for storing, retrieving, and managing document-oriented, or semi structured, information. Document-oriented databases are one of the main categories of NoSQL databases.
- An embedded database system is a DBMS which is tightly integrated with an application software that requires access to stored data in such a way that the DBMS is hidden from the application's end-users and requires little or no ongoing maintenance.[21]
- End-user databases consist of data developed by individual end-users. Examples of these are collections of documents, spreadsheets, presentations, multimedia, and other files. Several products[*which?*] exist to support such databases.
- A federated database system comprises several distinct databases, each with its own DBMS. It is handled as a single database by a federated database management system (FDBMS), which transparently integrates multiple autonomous DBMSs, possibly of different types (in which case it would also be a heterogeneous database system), and provides them with an integrated conceptual view.
- Sometimes the term *multi-database* is used as a synonym for federated database, though it may refer to a less integrated (e.g., without an FDBMS and a managed integrated schema) group of databases that cooperate in a single application. In this case, typically middleware is used for distribution, which typically includes an atomic commit protocol (ACP), e.g., the two-phase commit protocol, to allow distributed (global) transactions across the participating databases.
- A graph database is a kind of NoSQL database that uses graph structures with nodes, edges, and properties to represent and store information. General graph

databases that can store any graph are distinct from specialized graph databases such as triplestores and network databases.

- An array DBMS is a kind of NoSQL DBMS that allows modeling, storage, and retrieval of (usually large) multi-dimensional arrays such as satellite images and climate simulation output.
- In a hypertext or hypermedia database, any word or a piece of text representing an object, e.g., another piece of text, an article, a picture, or a film, can be hyperlinked to that object. Hypertext databases are particularly useful for organizing large amounts of disparate information. For example, they are useful for organizing online encyclopedias, where users can conveniently jump around the text. The World Wide Web is thus a large distributed hypertext database.
- A knowledge base (abbreviated **KB**, **kb** or Δ[22][23]) is a special kind of database for knowledge management, providing the means for the computerized collection, organization, and retrieval of knowledge. Also a collection of data representing problems with their solutions and related experiences.

- A mobile database can be carried on or synchronized from a mobile computing device.
- Operational databases store detailed data about the operations of an organization. They typically process relatively high volumes of updates using transactions. Examples include customer databases that record contact, credit, and demographic information about a business's customers, personnel databases that hold information such as salary, benefits, skills data about employees, enterprise resource planning systems that record details about product components, parts inventory, and financial databases that keep track of the organization's money, accounting and financial dealings.
- A parallel database seeks to improve performance through parallelization for tasks such as loading data, building indexes and evaluating queries.

  The major parallel DBMS architectures which are induced by the underlying hardware architecture are:

  - **Shared memory architecture**, where multiple processors share the main memory space, as well as other data storage.
  - **Shared disk architecture**, where each processing unit (typically consisting of multiple processors) has its own main memory, but all units share the other storage.
  - **Shared-nothing architecture**, where each processing unit has its own main memory and other storage.

- Probabilistic databases employ fuzzy logic to draw inferences from imprecise data.
- Real-time databases process transactions fast enough for the result to come back and be acted on right away.
- A spatial database can store the data with multidimensional features. The queries on such data include location-based queries, like "Where is the closest hotel in my area?".

- A temporal database has built-in time aspects, for example a temporal data model and a temporal version of SQL. More specifically the temporal aspects usually include valid-time and transaction-time.
- A terminology-oriented database builds upon an object-oriented database, often customized for a specific field.
- An unstructured data database is intended to store in a manageable and protected way diverse objects that do not fit naturally and conveniently in common databases. It may include email messages, documents, journals, multimedia objects, etc. The name may be misleading since some objects can be highly structured. However, the entire possible object collection does not fit into a predefined structured framework. Most established DBMSs now support unstructured data in various ways, and new dedicated DBMSs are emerging.

Database management system

Connolly and Begg define database management system (DBMS) as a "software system that enables users to define, create, maintain and control access to the database."[24] Examples of DBMS's include MySQL, MariaDB, PostgreSQL, Microsoft SQL Server, Oracle Database, and Microsoft Access.

The DBMS acronym is sometimes extended to indicate the underlying database model, with RDBMS for the relational, OODBMS for the object (oriented) and ORDBMS for the object–relational model. Other extensions can indicate some other characteristics, such as DDBMS for a distributed database management systems.

The functionality provided by a DBMS can vary enormously. The core functionality is the storage, retrieval and update of data. Codd proposed the following functions and services a fully-fledged general purpose DBMS should provide:[25]

- Data storage, retrieval and update
- User accessible catalog or data dictionary describing the metadata
- Support for transactions and concurrency
- Facilities for recovering the database should it become damaged
- Support for authorization of access and update of data
- Access support from remote locations
- Enforcing constraints to ensure data in the database abides by certain rules

It is also generally to be expected the DBMS will provide a set of utilities for such purposes as may be necessary to administer the database effectively, including import, export, monitoring, defragmentation and analysis utilities.[26] The core part of the DBMS interacting between the database and the application interface sometimes referred to as the database engine.

Often DBMSs will have configuration parameters that can be statically and dynamically tuned, for example the maximum amount of main memory on a server the database can use. The trend is to minimize the amount of manual configuration, and for cases such as embedded databases the need to target zero-administration is paramount.

The large major enterprise DBMSs have tended to increase in size and functionality and have involved up to thousands of human years of development effort throughout their lifetime.[a]

Early multi-user DBMS typically only allowed for the application to reside on the same computer with access via terminals or terminal emulation software. The client–server architecture was a development where the application resided on a client desktop and the database on a server allowing the processing to be distributed. This evolved into a multitier architecture incorporating application servers and web servers with the end user interface via a web browser with the database only directly connected to the adjacent tier.[28]

A general-purpose DBMS will provide public application programming interfaces (API) and optionally a processor for database languages such as SQL to allow applications to be written to interact with and manipulate the database. A special purpose DBMS may use a private API and be specifically customized and linked to a single application. For example, an email system performs many of the functions of a general-purpose DBMS such as message insertion, message deletion, attachment handling, blocklist lookup, associating messages an email address and so forth however these functions are limited to what is required to handle email.

Application

External interaction with the database will be via an application program that interfaces with the DBMS.[29] This can range from a database tool that allows users to execute SQL queries textually or graphically, to a website that happens to use a database to store and search information.

Application program interface

A programmer will code interactions to the database (sometimes referred to as a datasource) via an application program interface (API) or via a database language. The particular API or language chosen will need to be supported by DBMS, possibly indirectly via a preprocessor or a bridging API. Some API's aim to be database independent, ODBC being a commonly known example. Other common API's include JDBC and ADO.NET.

Database languages

Database languages are special-purpose languages, which allow one or more of the following tasks, sometimes distinguished as sublanguages:

- Data control language (DCL) – controls access to data;
- Data definition language (DDL) – defines data types such as creating, altering, or dropping tables and the relationships among them;
- Data manipulation language (DML) – performs tasks such as inserting, updating, or deleting data occurrences;

- Data query language (DQL) – allows searching for information and computing derived information.

Database languages are specific to a particular data model. Notable examples include:

- SQL combines the roles of data definition, data manipulation, and query in a single language. It was one of the first commercial languages for the relational model, although it departs in some respects from the relational model as described by Codd (for example, the rows and columns of a table can be ordered). SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. The standards have been regularly enhanced since and are supported (with varying degrees of conformance) by all mainstream commercial relational DBMSs.[30][31]
- OQL is an object model language standard (from the Object Data Management Group). It has influenced the design of some of the newer query languages like JDOQL and EJB QL.
- XQuery is a standard XML query language implemented by XML database systems such as MarkLogic and eXist, by relational databases with XML capability such as Oracle and Db2, and also by in-memory XML processors such as Saxon.
- SQL/XML combines XQuery with SQL.[32]

A database language may also incorporate features like:

- DBMS-specific configuration and storage engine management
- Computations to modify query results, like counting, summing, averaging, sorting, grouping, and cross-referencing
- Constraint enforcement (e.g. in an automotive database, only allowing one engine type per car)
- Application programming interface version of the query language, for programmer convenience

Storage

Database storage is the container of the physical materialization of a database. It comprises the *internal* (physical) *level* in the database architecture. It also contains all the information needed (e.g., metadata, "data about the data", and internal data structures) to reconstruct the *conceptual level* and *external level* from the internal level when needed. Databases as digital objects contain three layers of information which must be stored: the data, the structure, and the semantics. Proper storage of all three layers is needed for future preservation and longevity of the database.[33] Putting data into permanent storage is generally the responsibility of the database engine a.k.a. "storage engine". Though typically accessed by a DBMS through the underlying operating system (and often using the operating systems' file systems as intermediates for storage layout), storage properties and configuration settings are extremely important for the efficient operation of the DBMS, and thus

are closely maintained by database administrators. A DBMS, while in operation, always has its database residing in several types of storage (e.g., memory and external storage). The database data and the additional needed information, possibly in very large amounts, are coded into bits. Data typically reside in the storage in structures that look completely different from the way the data look at the conceptual and external levels, but in ways that attempt to optimize (the best possible) these levels' reconstruction when needed by users and programs, as well as for computing additional types of needed information from the data (e.g., when querying the database).

Some DBMSs support specifying which character encoding was used to store data, so multiple encodings can be used in the same database.

Various low-level database storage structures are used by the storage engine to serialize the data model so it can be written to the medium of choice. Techniques such as indexing may be used to improve performance. Conventional storage is row-oriented, but there are also column-oriented and correlation databases.

Materialized views

Often storage redundancy is employed to increase performance. A common example is storing *materialized views*, which consist of frequently needed *external views* or query results. Storing such views saves the expensive computing them each time they are needed. The downsides of materialized views are the overhead incurred when updating them to keep them synchronized with their original updated database data, and the cost of storage redundancy.

Replication

Occasionally a database employs storage redundancy by database objects replication (with one or more copies) to increase data availability (both to improve performance of simultaneous multiple end-user accesses to the same database object, and to provide resiliency in a case of partial failure of a distributed database). Updates of a replicated object need to be synchronized across the object copies. In many cases, the entire database is replicated.

Virtualization

With data virtualization, the data used remains in its original locations and real-time access is established to allow analytics across multiple sources. This can aid in resolving some technical difficulties such as compatibility problems when combining data from various platforms, lowering the risk of error caused by faulty data, and guaranteeing that the newest data is used. Furthermore, avoiding the creation of a new database containing personal information can make it easier to comply with privacy regulations. However, with data virtualization, the connection to all necessary data sources must be operational as there is no local copy of the data, which is one of the main drawbacks of the approach.[34]