DATABASE

Organized collection of data in computing

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.

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

I. ***Data definition – Creation, modification and removal of definitions that detail how the data is to be organized.***
II. ***Update – Insertion, modification, and deletion of the data itself.***
III. ***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.***
IV. ***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.***

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

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.[citation needed]

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

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, 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.

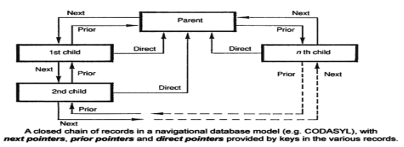
The relational model, first proposed in 1970 by Edgar F. Codd, departed from this tradition by insisting that applications should search for data by content, rather than by following links. The relational model employs sets of ledger-style tables, each used for a different type of entity. Only in the 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. The dominant database language, standardized SQL for the relational model, has influenced database languages for other data models.

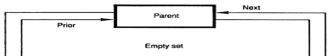
Object databases were developed in the 1980s to overcome the inconvenience of object–relational impedance mismatch, which led to the coining of the term "post-relational" and also the development of hybrid object–relational databases.

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





Database , Data hierarchy , Cloud Database

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.

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:

- *Use of a primary key (known as a CALC key, typically implemented by hashing)*
- *Navigating relationships (called sets) from one record to another*
- *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.

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.

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.

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 based on D.L. Childs' Set-Theoretic Data model. 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. The system remained in production until 1998.

Integrated approach

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.

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." 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. 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

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.

Use cases

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.

1. *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.*
2. *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.*

3. *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.*
4. *Data warehouses 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.*
5. *A deductive database combines logic programming with a relational database.*
6. *A distributed database is one in which both the data and the DBMS span multiple computers.*
7. *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.*
8. *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.*
9. *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.*
10. *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.*
11. *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.*
12. *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.*
13. *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.*
14. *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.*
15. *A knowledge base (abbreviated KB, kb or Δ) 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.*
16. *A mobile database can be carried on or synchronized from a mobile computing device.*
17. *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.

18. *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:
    a. *Shared memory architecture, where multiple processors share the main memory space, as well as other data storage.*
    b. *Shared disk architecture, where each processing unit (typically consisting of multiple processors) has its own main memory, but all units share the other storage.*
    c. *Shared-nothing architecture, where each processing unit has its own main memory and other storage.*
19. *Probabilistic databases employ fuzzy logic to draw inferences from imprecise data.*
20. *Real-time databases process transactions fast enough for the result to come back and be acted on right away.*
21. *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?".*
22. *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.*
23. *A terminology-oriented database builds upon an object-oriented database, often customized for a specific field.*
24. *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." 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:

1. *Data storage, retrieval and update*
2. *User accessible catalog or data dictionary describing the metadata*
3. *Support for transactions and concurrency*
4. *Facilities for recovering the database should it become damaged*
5. *Support for authorization of access and update of data*
6. *Access support from remote locations*
7. *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. 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.

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.

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. 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:

1. *Data control language (DCL) – controls access to data;*
2. *Data definition language (DDL) – defines data types such as creating, altering, or dropping tables and the relationships among them;*
3. *Data manipulation language (DML) – performs tasks such as inserting, updating, or deleting data occurrences;*
4. *Data query language (DQL) – allows searching for information and computing derived information.*

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

1. *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.*
2. *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.*
3. *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.*
4. *SQL/XML combines XQuery with SQL.*

A database language may also incorporate features like:

1. *DBMS-specific configuration and storage engine management*
2. *Computations to modify query results, like counting, summing, averaging, sorting, grouping, and cross-referencing*
3. *Constraint enforcement (e.g. in an automotive database, only allowing one engine type per car)*
4. *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. 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.

Security

Database security deals with all various aspects of protecting the database content, its owners, and its users. It ranges from protection from intentional unauthorized database uses to unintentional database accesses by unauthorized entities (e.g., a person or a computer program).

Database access control deals with controlling who (a person or a certain computer program) are allowed to access what information in the database. The information may comprise specific database objects (e.g., record types, specific records, data structures), certain computations over certain objects (e.g., query types, or specific queries), or using specific access paths to the former (e.g., using specific indexes or other data structures to access information). Database access controls are set by special authorized (by the database owner) personnel that uses dedicated protected security DBMS interfaces.

This may be managed directly on an individual basis, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called "subschemas". For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data. If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases.

Data security in general deals with protecting specific chunks of data, both physically (i.e., from corruption, or destruction, or removal; e.g., see physical security), or the interpretation of them, or parts of them to meaningful information (e.g., by looking at the strings of bits that they comprise, concluding specific valid credit-card numbers; e.g., see data encryption).

Change and access logging records who accessed which attributes, what was changed, and when it was changed. Logging services allow for a forensic database audit later by keeping a record of access occurrences and changes. Sometimes application-level code is used to record changes rather than leaving this in the database. Monitoring can be set up to attempt to detect security breaches. Therefore, organizations must take database security seriously because of the many benefits it provides. Organizations will be safeguarded from security breaches and hacking activities like

firewall intrusion, virus spread, and ransom ware. This helps in protecting the company's essential information, which cannot be shared with outsiders at any cause.

Transactions and concurrency

Database transactions can be used to introduce some level of fault tolerance and data integrity after recovery from a crash. A database transaction is a unit of work, typically encapsulating a number of operations over a database (e.g., reading a database object, writing, acquiring or releasing a lock, etc.), an abstraction supported in database and also other systems. Each transaction has well defined boundaries in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands).

The acronym ACID describes some ideal properties of a database transaction: atomicity, consistency, isolation, and durability.

Migration

A database built with one DBMS is not portable to another DBMS (i.e., the other DBMS cannot run it). However, in some situations, it is desirable to migrate a database from one DBMS to another. The reasons are primarily economical (different DBMSs may have different total costs of ownership or TCOs), functional, and operational (different DBMSs may have different capabilities). The migration involves the database's transformation from one DBMS type to another. The transformation should maintain (if possible) the database related application (i.e., all related application programs) intact. Thus, the database's conceptual and external architectural levels should be maintained in the transformation. It may be desired that also some aspects of the architecture internal level are maintained. A complex or large database migration may be a complicated and costly (one-time) project by itself, which should be factored into the decision to migrate. This is in spite of the fact that tools may exist to help migration between specific DBMS interfaces that support bulk insertion) before making it operational. In some cases, the database becomes operational while empty of application data, and data are accumulated during its operation.

After the database is created, initialized and populated it needs to be maintained. Various database parameters may need changing and the database may need to be tuned (tuning) for better performance; application's data structures may be changed or added, new related application programs may be written to add to the application's functionality, etc.

Backup and restore

Backup

Sometimes it is desired to bring a database back to a previous state (for many reasons, e.g., cases when the database is found corrupted due to a software error, or if it has been updated with erroneous data). To achieve this, a backup operation is done occasionally or continuously, where each desired database state (i.e., the values of its data and their embedding in database's data structures) is kept within dedicated backup files (many techniques exist to do this effectively). When it is decided by a database administrator to bring the database back to this state (e.g., by specifying this state by a desired point in time when the database was in this state), these files are used to restore that state.

Static analysis

*Static analysis techniques for software verification can be applied also in the scenario of query languages. In particular, the *Abstract interpretation framework has been extended to the field of query languages for relational databases as a way to support sound approximation techniques. The semantics of query languages can be tuned according to suitable abstractions of the concrete domain of data. The abstraction of relational database systems has many interesting applications, in particular, for security purposes, such as fine-grained access control, watermarking, etc.*

Miscellaneous features

Other DBMS features might include:

1. *Database logs – This helps in keeping a history of the executed functions.*
2. *Graphics component for producing graphs and charts, especially in a data warehouse system.*
3. *Query optimizer – Performs query optimization on every query to choose an efficient query plan (a partial order (tree) of operations) to be executed to compute the query result. May be specific to a particular storage engine.*
4. *Tools or hooks for database design, application programming, application program maintenance, database performance analysis and monitoring, database configuration monitoring, DBMS hardware configuration (a DBMS and related database may span computers, networks, and storage units) and related database mapping (especially for a distributed DBMS), storage allocation and database layout monitoring, storage migration, etc.*

Increasingly, there are calls for a single system that incorporates all of these core functionalities into the same build, test, and deployment framework for database management and source control. Borrowing from other developments in the software industry, some market such offerings as "DevOps for database".

Design and modeling

The first task of a database designer is to produce a conceptual data model that reflects the structure of the information to be held in the database. A common approach to this is to develop an entity–relationship model, often with the aid of drawing tools. Another popular approach is the Unified Modeling Language. A successful data model will accurately reflect the possible state of the external world being modeled: for example, if people can have more than one phone number, it will allow this information to be captured. Designing a good conceptual data model requires a good understanding of the application domain; it typically involves asking deep questions about the things of interest to an organization, like "can a customer also be a supplier?", or "if a product is sold with two different forms of packaging, are those the same product or different products?", or "if a plane flies from New York to Dubai via Frankfurt, is that one flight or two (or maybe even three)?". The answers to these questions establish definitions of the terminology used for entities (customers, products, flights, flight segments) and their relationships and attributes.

Producing the conceptual data model sometimes involves input from business processes, or the analysis of workflow in the organization. This can help to establish what information is needed in the database, and what can be left out. For example, it can help when deciding whether the database needs to hold historic data as well as current data.

Having produced a conceptual data model that users are happy with, the next stage is to translate this into a schema that implements the relevant data structures within the database. This process is often called logical database design, and the output is a logical data model expressed in the form of a schema. Whereas the conceptual data model is (in theory at least) independent of the choice of database technology, the logical data model will be expressed in terms of a particular database model supported by the chosen DBMS. (The terms data model and database model are often used interchangeably, but in this article we use data model for the design of a specific database, and database model for the modeling notation used to express that design).

The most popular database model for general-purpose databases is the relational model, or more precisely, the relational model as represented by the SQL language. The process of creating a logical database design using this model uses a methodical approach known as normalization. The goal of normalization is to ensure that each elementary "fact" is only recorded in one place, so that insertions, updates, and deletions automatically maintain consistency.

The final stage of database design is to make the decisions that affect performance, scalability, recovery, security, and the like, which depend on the particular DBMS. This is often called physical database design, and the output is the physical data model. A key goal during this stage is data independence, meaning that the decisions made for performance optimization purposes should be invisible to end-users and applications. There are two types of data independence: Physical data independence and logical data independence. Physical design is driven mainly by performance requirements, and requires a good knowledge of the expected workload and access patterns, and a deep understanding of the features offered by the chosen DBMS.

Another aspect of physical database design is security. It involves both defining access control to database objects as well as defining security levels and methods for the data itself.

Models

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model (or the SQL approximation of relational), which uses a table-based format.

Common logical data models for databases include:

1. *Navigational databases*
    a. *Hierarchical database model*
    b. *Network model*
    c. *Graph database*
2. *Relational model*
3. *Entity–relationship model*
    a. *Enhanced entity–relationship model*
4. *Object model*
5. *Document model*
6. *Entity–attribute–value model*
7. *Star schema*

An object–relational database combines the two related structures.

Physical data models include:

1. *Inverted index*
2. *Flat file*

Other models include:

1. *Multidimensional model*
2. *Array model*
3. *Multivalue model*

Specialized models are optimized for particular types of data:

1. *XML database*
2. *Semantic model*
3. *Content store*
4. *Event store*
5. *Time series model*

External, conceptual, and internal views

A database management system provides three views of the database data:

1. *The external level defines how each group of end-users sees the organization of data in the database. A single database can have any number of views at the external level.*
2. *The conceptual level (or logical level) unifies the various external views into a compatible global view. It provides the synthesis of all the external views. It is out of the scope of the*

various database end-users, and is rather of interest to database application developers and database administrators.

3. *The internal level (or physical level) is the internal organization of data inside a DBMS. It is concerned with cost, performance, scalability and other operational matters. It deals with storage layout of the data, using storage structures such as indexes to enhance performance. Occasionally it stores data of individual views (materialized views), computed from generic data, if performance justification exists for such redundancy. It balances all the external views' performance requirements, possibly conflicting, in an attempt to optimize overall performance across all activities.*

While there is typically only one conceptual and internal view of the data, there can be any number of different external views. This allows users to see database information in a more business-related way rather than from a technical, processing viewpoint. For example, a financial department of a company needs the payment details of all employees as part of the company's expenses, but does not need details about employees that are in the interest of the human resources department. Thus different departments need different views of the company's database.

The three-level database architecture relates to the concept of data independence which was one of the major initial driving forces of the relational model. The idea is that changes made at a certain level do not affect the view at a higher level. For example, changes in the internal level do not affect application programs written using conceptual level interfaces, which reduces the impact of making physical changes to improve performance.

The conceptual view provides a level of indirection between internal and external. On the one hand it provides a common view of the database, independent of different external view structures, and on the other hand it abstracts away details of how the data are stored or managed (internal level). In principle every level, and even every external view, can be presented by a different data model. In practice usually a given DBMS uses the same data model for both the external and the conceptual levels (e.g., relational model). The internal level, which is hidden inside the DBMS and depends on its implementation, requires a different level of detail and uses its own types of data structure types.

Research

Database technology has been an active research topic since the 1960s, both in academia and in the research and development groups of companies (for example IBM Research). Research activity includes theory and development of prototypes. Notable research topics have included models, the atomic transaction concept, related concurrency control techniques, query languages and query optimization methods, RAID, and more.

The database research area has several dedicated academic journals (for example, ACM Transactions on Database Systems-TODS, Data and Knowledge Engineering-DKE) and annual conferences (e.g., ACM SIGMOD, ACM PODS, VLDB, IEEE ICDE).

Related Extensions

- *I. Comparison of database tools*
- *II. Comparison of object database management systems*
- *III. Comparison of object–relational database management systems*
- *IV. Comparison of relational database management systems*
- *V. Data hierarchy*
- *VI. Data bank*
- *VII. Data store*
- *VIII. Database theory*
- *IX. Database testing*
- *X. Database-centric architecture*
- *XI. Datalog*
- *XII. Database-as-IPC*

XIII. DBOS
XIV. Flat-file database
XV. INP (database)
XVI. Journal of Database Management
XVII. Question-focused dataset

DATA HIERARCHY

Data hierarchy refers to the systematic organization of data, often in a hierarchical form. Data organization involves characters, fields, records, files and so on. This concept is a starting point when trying to see what makes up data and whether data has a structure. For example, how does a person make sense of data such as 'employee', 'name', 'department', 'Marcy Smith', 'Sales Department' and so on, assuming that they are all related? One way to understand them is to see these terms as smaller or larger components in a hierarchy. One might say that Marcy Smith is one of the employees in the Sales Department, or an example of an employee in that Department. The data we want to capture about all our employees, and not just Marcy, is the name, ID number, address etc.

Purpose of the data hierarchy

"Data hierarchy" is a basic concept in data and database theory and helps to show the relationships between smaller and larger components in a database or data file. It is used to give a better sense of understanding about the components of data and how they are related.

It is particularly important in databases with referential integrity, third normal form, or perfect key. "Data hierarchy" is the result of proper arrangement of data without redundancy. Avoiding redundancy eventually leads to proper "data hierarchy" representing the relationship between data, and revealing its relational structure.

Components of the data hierarchy

The components of the data hierarchy are listed below.

A data field holds a single fact or attribute of an entity. Consider a date field, e.g. "19 September 2004". This can be treated as a single date field (e.g. birthdate), or three fields, namely, day of month, month and year.

A record is a collection of related fields. An Employee record may contain a name field(s), address fields, birthdate field and so on.

A file is a collection of related records. If there are 100 employees, then each employee would have a record (e.g. called Employee Personal Details record) and the collection of 100 such records would constitute a file (in this case, called Employee Personal Details file).

Files are integrated into a database. This is done using a Database Management System. If there are other facets of employee data that we wish to capture, then other files such as Employee Training History file and Employee Work History file could be created as well.

Illustration of the data hierarchy

An illustration of the above description is shown in this diagram below:

<table>
<tr><th>Hierarchy</th><th>Example</th></tr>
<tr><td>Database</td><td>Employee Database

Employee Details File
Training Records File
Salary File</td></tr>
<tr><td>File</td><td>Employee Details File

EMP_NAME | JOB TITLE | DATE EMPLOYED
Alice Carter | Lecturer | 31 Mar 2002
Faridah bte Hassan | Sales Manager | 9 Aug 2013
Jeffrey Tan | Lecturer | 19 Sep 2004
Steve Willis | HR Manager | 23 Dec 2005</td></tr>
<tr><td>Record</td><td>Employee Record

EMP_NAME | JOB TITLE | DATE EMPLOYED
Jeffrey Tan | Lecturer | 19 Sep 2004</td></tr>
<tr><td>Field</td><td>Employee Name Field

EMP_NAME
Jeffrey Tan</td></tr>
<tr><td>Byte</td><td>01001010 (Letter J in ASCII)</td></tr>
<tr><td>Bit</td><td>0</td></tr>
</table>

Note: EMP = employee

Source: Jeffrey TL Tan Wikipedia original contributor for Data Hierarchy. 9 Aug 2013
Permission is given to freely use this diagram in its entirety & unedited.

Data Hierarchy Diagram – with Employee Database example

Data Hierarchy diagram
The following terms are for better clarity. With reference to the example in the above diagram:

Data field label = Employee Name or EMP_NAME

Data field value = Jeffrey Tan

The above description is a view of data as understood by a user e.g. a person working in Human Resource Department.

The above structure can be seen in the hierarchical model, which is one way to organize data in a database.

In terms of data storage, data fields are made of bytes and these in turn are made up of bits.

Related Extensions

I. ***Data dictionary – Set of metadata that contains definitions and representations of data elements***
II. ***Data element – Semantic representation of data***
III. ***Data acquisition – Process of sampling signals from sensors and converting into digital data***

CLOUD DATABASE

Database running on a cloud computing platform

A cloud database is a database that typically runs on a cloud computing platform and access to the database is provided as-a-service. There are two common deployment models: users can run databases on the cloud independently, using a virtual machine image, or they can purchase access to a database service, maintained by a cloud database provider. Of the databases available on the cloud, some are SQL-based and some use a NoSQL data model.

Database services take care of scalability and high availability of the database. Database services make the underlying software-stack transparent to the user.

Deployment models

There are two primary methods to run a database on a cloud platform:
Cloud platforms allow users to purchase virtual-machine instances for a limited time, and one can run a database on such virtual machines. Users can either upload their own machine image with a database installed on it, or use ready-made machine images that already include an optimized installation of a database.

Database-as-a-service (DBaaS)

With a database as a service model, users pay fees to a cloud provider for services and computing resources, reducing the amount of money and effort needed to develop and manage databases. Users are given tools to create and manage database instances, and control users. Some cloud providers also offer tools to manage database structures and data. Many cloud providers offer both relational (Amazon RDS, SQL Server) and NoSQL (MongoDB, Amazon DynamoDB) databases. This is a type of software as a service (SaaS).

Architecture and common characteristics

Most database services offer web-based consoles, which the end user can use to provision and configure database instances.

Database services consist of a database-manager component, which controls the underlying database instances using a service API. The service API is exposed to the end user, and permits users to perform maintenance and scaling operations on their database instances.
Underlying software-stack stack typically includes the operating system, the database and third-party software used to manage the database. The service provider is responsible for installing, patching and updating the underlying software stack and ensuring the overall health and performance of the database.

Scalability features differ between vendors – some offer auto-scaling, others enable the user to scale up using an API, but do not scale automatically.
There is typically a commitment for a certain level of high availability (e.g. 99.9% or 99.99%). This is achieved by replicating data and failing instances over to other database instances.

Data model

The design and development of typical systems utilize data management and relational databases as their key building blocks. Advanced queries expressed in SQL work well with the strict relationships that are imposed on information by relational databases. However, relational database technology was not initially designed or developed for use over distributed systems. This issue has been addressed with the addition of clustering enhancements to the relational databases, although some basic tasks require complex and expensive protocols, such as with data synchronization.

Modern relational databases have shown poor performance on data-intensive systems, therefore, the idea of NoSQL has been utilized within database management systems for cloud based systems.

Within NoSQL implemented storage, there are no requirements for fixed table schemas, and the use of join operations is avoided. "The NoSQL databases have proven to provide efficient horizontal scalability, good performance, and ease of assembly into cloud applications." Data models relying on simplified relay algorithms have also been employed in data-intensive cloud mapping applications unique to virtual frameworks.

It is also important to differentiate between cloud databases which are relational as opposed to non-relational or NoSQL:

SQL databases

SQL databases are one type of database which can run in the cloud, either in a virtual machine or as a service, depending on the vendor. While SQL databases are easily vertically scalable, horizontal scalability poses a challenge, that cloud database services based on SQL have started to address.

NoSQL databases

NoSQL databases are another type of database which can run in the cloud. NoSQL databases are built to service heavy read/write loads and can scale up and down easily, and therefore they are more natively suited to running in the cloud. However, most contemporary applications are built around an SQL data model, so working with NoSQL databases often requires a complete rewrite of application code.

Some SQL databases have developed NoSQL capabilities including JSON, binary JSON (e.g. BSON or similar variants), and key-value store data types.
A multi-model database with relational and non-relational capabilities provides a standard SQL interface to users and applications and thus facilitates the usage of such databases for contemporary applications built around an SQL data model. Native multi-model databases support multiple data models with one core and a unified query language to access all data models.

Vendors

The following table lists notable database vendors with a cloud database offering, classified by their deployment model – machine image vs. database as a service – and data model, SQL vs. NoSQL.

More information Virtual Machine Deployment, Database as a Service ...

Related Extensions

- *Cloud computing*
- *Cloud storage*
- *Data as a service*
- *Relational database*