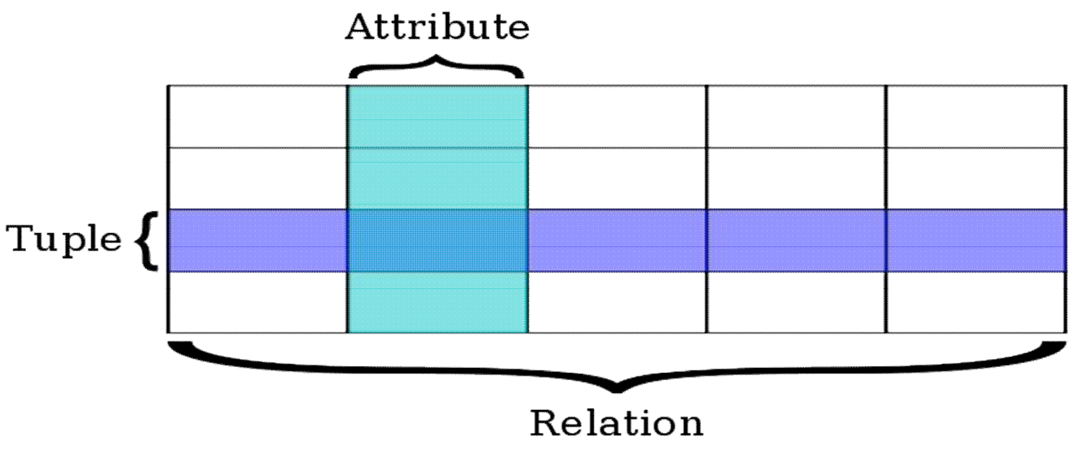
Database

From Wikipedia



2014

대구대 사회과학대학 문헌정보학과

DEFINITION

A database is an organized collection of data. The data are typically organized to model relevant aspects of reality in a way that supports processes requiring this information. For example, modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

데이터베이스란 데이터를 조직적으로 모아놓은 것이다. 데이터는 전형적으로 이 정보를 필요로 하는 처리과정을 지원하는 방식으로 적절한 모습의 실체를 모델하기 위하여 조직된다.

Database management systems (DBMSs) are specially designed applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose database management system (DBMS) is a software system designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, MariaDB, PostgreSQL, SQLite, Microsoft SQL Server, Oracle,SAP, dBASE, FoxPro, IBM DB2, LibreOffice Base and FileMaker Pro. A database is not generally portable across different DBMS, but different DBMSs can interoperate by using standards such as SQL and ODBC or JDBC to allow a single application to work with more than one database.

DBMS는 데이터를 수집하고 분석하기 위하여 특히 이용자, 또 다른 어플, 그리고 데이터베이스 그 자체와 상호작용하기 위한 어플을 디자인 한다. 일반용인 DBMSs는 하나의 소프트웨어 시스템이며, 이것은 데이터베이스의 정의, 제작, 질문, 갱신, 그리고 운영이 가능하도록 디자인되었다. 잘 알려진 DBMSs에는 MySQL, MariaDB, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, SAP, dBASE, FoxPro, IBM DB2, LibreOffice Base and FileMaker Pro가 있다. 데이터베이스는 서로 다른 DBAMSs로 이동할 수 없는 것이 일반적이지만 서로 다른 DBMSs는 하나의 어플로 하여금 하나 이상의 데이터베이스와 작업할 수 있도록 SQL과 ODBC 또는 JDBC과 같은 표준을 사용하여 서로 간에 운영이 가능하다.

1) SQL(Structured Query Language) is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS).
2) ODBC(Open Database Connectivity) is a standard programming language middleware API for accessing database management systems (DBMS).
3) JDBC is a Java-based data access technology (Java Standard Edition platform) from Oracle Corporation. This technology is an API for the Java programming language that defines how a client may access a database.
4) Application Programming Interface (API) specifies how some software components should interact with each other. In addition to accessing databases or computer hardware, such as hard disk drives or video cards, an API can be used to ease the work of programming graphical user interface components.

Contents

1. Terminology and overview

Formally, "database" refers to the data itself and supporting data structures.

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

데이터베이스는 정보의 입력, 저장, 검색, 관리를 위하여 대량의 정보를 운영할 수 있도록 제작된다. 데이터베이스는 한 세트의 소프트웨어 프로그램을 통하여 모든 이용자가 모든 데이터에 접근할 수 있도록 설치된다.

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

DBMS란 한 벌의 컴퓨터 소프트웨어이며 이용자와 하나의 데이터베이스 또는 복수의 데이터베이스와의 접속을 제공한다. 이것들이 매우 밀접하게 서로 관련되어 있기 때문에 “데이터베이스”란 용어는 아무 생각 없이 사용될 때, 종종 DBMS 그리고 그것이 운영하는 데이터 둘 다를 표현되기도 한다.

Outside the world of professional information technology, the term database is sometimes used casually to refer to any collection of data (perhaps a spreadsheet, maybe even a card index). The interactions catered for by most existing DBMS fall into four main groups:

Data definition. Defining new data structures for a database, removing data structures from the database, modifying the structure of existing data.
Update. Inserting, modifying, and deleting data.
Retrieval. Obtaining information either for end-user queries and reports or for processing by applications.

Administration. Registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control, and recovering information if the system fails.

전문적인 정보기술 분야를 벗어나서, 데이터베이스란 용어는 때때로 종종 어떤 데이터의 모음을 말하기도 한다(아마도 스프레드시트, 또는 카드색인까지도). 대부분의 기존의 DBMS에 의해 제공되는 상호작용은 다음과 같이 4가지로 나눌 수 있다:

1) 데이터 정의.
   데이터베이스를 위한 새로운 데이터 구조를 정의하고, 데이터베이스로부터 데이터 구조를 제거하고, 기존 데이터의 구조를 변경하는 것.
2) 갱신.
   데이터의 입력, 변경, 삭제
3) 검색.
   엔드유저의 쿼리와 리포트 또는 어플의 처리를 위한 정보의 획득
4) 관리
   이용자의 등록과 감시, 데이터 보안 강화, 성능 감시, 데이터의 순수성 유지, 병행통제 처리, 시스템 문제시 정보의 회복.

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

DBMS는 저장된 데이터의 순수성과 안전성을 유지하고, 시스템이 잘못될 때 정보를 회복할 책임이 있다.

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.

데이터베이스와 그것의 DBMS 둘 다 특별한 데이터베이스 모델의 원칙에 따라야 한다. “데이터베이스 시스템”이란 데이터베이스 모델, 데이터베이스 관리 시스템, 그리고 데이터베이스를 집합적으로 언급한 것이다.

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. RAID is used for recovery of data if any of the disks fails. 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 economical market, computer and storage vendors often take into account DBMS requirements in their own development plans.

물리적으로 말해서, 데이터베이스 서버는 전용 컴퓨터들이며 실제적인 데이터베이스를 보유하고 있고 단지 DBMS와 관련 소프트웨어를 가동하고 있다. 데이터베이스 서버는 대부분이 멀티프로세서 컴퓨터들이며, 안정된 저장을 위해 풍부한 메모리와 RAID 디스크 어레이를 갖고 있다. RAID는 어떤 디스크가 깨졌을 때 데이터의 복원을 위해 사용된다. 하드웨어 데이터베이스 가속기 - 고속의 채널을 통해 하나이상의 서버에 연결되어 있음 - 또한 대량의 트랜

젝션 처리 환경에서 사용된다. DBMSs는 대부분의 데이터베이스 어플의 한 복판에 있다. DBMSs는 내장되어있으면서 네트워크 지원이 가능한 이용자의 다업무처리용 커널과 관련해서 구축될 수도 있지만, 현대의 DBMS는 전형적으로 이러한 기능을 제공하기 위하여 표준 운영체제에 의존하고 있다. DBMSs가 중요한 경제 시장을 구성하면서부터, 컴퓨터와 저장매체 상인들은 종종 그들 자신의 발전계획에 DBMS의 필요성을 고려하고 있다.

1) RAID is a storage technology that combines multiple disk drive components into a logical unit for the purposes of data redundancy and performance improvement.
2) the kernel is a computer program that manages input/output requests from software and translates them into data processing instructions for the central processing unit and other electronic components of a computer.

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.

데이터베이스와 DBMSs는 그것들이 지원하는 데이터베이스 모델, 그것들이 기동되는 컴퓨터의 종류, 데이터베이스의 접근을 위해 사용되는 쿼리 언어, 그리고 성능, 규모, 복원, 그리고 안전에 영향을 끼치는 그것들의 내부 엔지니어링에 따라 범주화할 수 있다.

2. Applications and roles

Most organizations in developed countries today depend on databases for their business operations. Increasingly, databases are not only used to support the internal operations of the organization, but also to underpin its online interactions with customers and suppliers (see Enterprise software). Databases are not used only to hold administrative information, but are often embedded within applications to hold more specialized data: for example engineering data or economic models. Examples of database applications include computerized library systems, flight reservation systems, and computerized parts inventory systems.

오늘날 선진국의 대부분 기관들은 자신들의 업무활동을 위해 데이터베이스에 의존하고 있다. 점차적으로, 데이터베이스들은 그 기관의 내부업무를 지원하는데 사용될 뿐만 아니라 고객과 납품업자와의 온라인 상회작용에 있어서 중요하게 사용되고 있다. 데이터베이스는 관리정보를 유지하기 위하여 사용되는 것이 아니라 종종 더욱 전문화된 데이터를 보관하기 위한 어플 속에 내장된다. 데이터베이스 어플의 예로는 전산화된 도서관 시스템, 항공예약시스템, 그리고 전산화된 parts inventory systems(An inventory control system is a process for managing and locating objects or materials. Modern inventory control systems often rely upon barcodes and radio-frequency identification (RFID) tags to provide automatic identification of inventory objects)가 있다.

Client-server or transactional DBMSs are often complex to maintain high performance, availability and security when many users are querying and updating the database at the same time. Personal, desktop-based database systems tend to be less complex. For example, FileMaker and Microsoft Access come with built-in graphical user interfaces.

클라이언트-서버 또는 트랜젝션 DBMSs는 많은 이용자가 동시에 그 데이터베이스에 질문하고 갱신할 때 높은 성능, 이용성, 보안성을 유지하는 것이 종종 복잡해진다. 개인용이면서 데스크 탑인 데이터베이스 시스템은 복잡성이 다소 떨어지는 경향이 있다. 예를 들어, FileMaker와 Microsoft Access는 내장된 GUI를 사용하여 이용할 수 있다.

2.1 General-purpose and special-purpose DBMSs

A DBMS has evolved into a complex software system and its development typically requires thousands of person-years of development effort.[4] Some general-purpose DBMSs such as Adabas, Oracle and DB2 have been undergoing upgrades since the 1970s. General-purpose DBMSs aim to meet the needs of as many applications as possible, which adds to the complexity. However, the fact that their development cost can be spread over a large number of users means that they are often the most cost-effective approach.

DBMS는 하나의 복잡한 소프트웨어 시스템으로 진화하였으며 그것의 발전은 전형적으로 수많은 사람과 오랜 시간의 개발 노력을 요구하고 있다. Adabas, Oracle, DB2와 같은 범용의 DBMSs는 1970년대부터 지속적으로 업데이트가 이루어지고 있다. 범용의 DBMSs는 복잡성을 추가하여 가능한 한 많은 어플의 요구를 충족시키는 것을 목적으로 하고 있다. 그렇지만, 이것들의 개발 비용이 수많은 이용자에게 확산된다는 사실은 이것들이 가장 비용 대 효과를 고려해야 한다는 것을 의미하는 것이다.

However, a general-purpose DBMS is not always the optimal solution: in some cases a general-purpose DBMS may introduce unnecessary overhead. Therefore, there are many examples of systems that use special-purpose databases. A common example is an email system: email systems are designed to optimize the handling of email messages, and do not need significant portions of a general-purpose DBMS functionality.

그렇지만, 범용 DBMS가 항상 최적의 해결책은 아니다: 어떤 경우에 있어서, 범용 DBMS

는 불필요한 overhead를 불러올지도 모른다. 그러므로 특별한 목적용의 데이터베이스를 사용해야하는 시스템에 대한 많은 예가 존재한다. 한 가지 일반적인 예는 이메일시스템이다: 이메일시스템은 이메일 메시지를 최대로 잘 처리하도록 디자인되었으며 범용 DBMS의 기능성을 크게 필요로 하진 않는다.

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

많은 데이터베이스가 최종 이용자를 대신하여 직접적으로 DBMS 인터페이스를 사용하지 않고 그 데이터베이스에 접근할 수 있는 어플 소프트웨어를 가지고 있다. 어플 프로그래머는 직접적으로 또는 wire protocol을 사용할 수도 있지만 api를 통해 더 많이 사용할 수도 있다. 데이터베이스 디자이너와 데이터베이스 운영자는 어플 데이터베이스의 구축과 유지를 위하여 전용 인터페이스를 통하여 DBMS와 상호작용한다. 따라서 그들은 DBMS의 운영 방법과 DBMS의 외부 인터페이스와 조절 요소에 대하여 더 많은 지식과 이해력을 필요로 한다.

1) a wire protocol refers to a way of getting data from point to point: A wire protocol is needed if more than one application has to interoperate. In contrast to transport protocols at the transport level (like TCP or UDP), the term "wire protocol" is used to describe a common way to represent information at the application level.

2) the transport layer or layer 4 provides end-to-end communication services for applications] within a layered architecture of network components and protocols. The transport layer provides convenient services such as connection-oriented data stream support, reliability, flow control, and multiplexing. Transport layers are contained in both the TCP/IP model (RFC 1122), [2] which is the foundation of the Internet, and the Open Systems Interconnection (OSI) model of general networking.

The definitions of the transport layer are slightly different in these two models. This article primarily refers to the TCP/IP model, in which TCP is largely for a convenient application programming interface to internet hosts, as opposed to the OSI-model definition of the transport layer.

The most well-known transport protocol is the Transmission Control Protocol

(TCP). It lent its name to the title of the entire Internet Protocol Suite, TCP/IP. It is used for connection-oriented transmissions, whereas the connectionless User Datagram Protocol (UDP) is used for simpler messaging transmissions. TCP is the more complex protocol, due to its stateful design incorporating reliable transmission and data stream services. Other prominent protocols in this group are the Datagram Congestion Control Protocol (DCCP) and the Stream Control Transmission Protocol (SCTP).

General-purpose databases are usually developed by one organization or community of programmers, while a different group builds the applications that use it. In many companies, specialized database administrators maintain databases, run reports, and may work on code that runs on the databases themselves (rather than in the client application).

범용 데이터베이스는 대부분이 어떤 기관이나 프로그래머 집단에 의해 개발되었다. 반면에 다양한 그룹에서 그것을 사용하기 위한 어플을 구축하고 있다. 많은 회사에서, 전문 데이터베이스 운영자가 데이터베이스를 관리하고, 보고서를 작성하고 있으며 클라이언트 어플에서라기 보다는 데이터베이스 그 자체를 기동시키는 코드를 가지고 작업하기도 한다.

3. History

With the data progress in technology in the areas of processors, computer memory, computer storage and computer networks, the sizes, capabilities, and performance of databases and their respective DBMSs have grown in orders of magnitudes. The 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, epitomized by IBM's IMS system, and the Codasyl model (Network model), implemented in a number of products such as IDMS.

프로세서 분야에서 데이터 처리기술의 진보와 더불어, 컴퓨터 메모리, 컴퓨터 메모리, 컴퓨터 네트워크, 데이터베이스의 크기, 용량, 성능 그리고 그것들을 다루는 각각의 DBMSs가 규모면에서 크게 성장하고 있다. 데이터베이스 기술의 발전은 데이터 모델이나 구조에 따라 3세대로 구분할 수 있다: navigational, SQL/relational, post-relational. 두 가지 주요한 초기 네비게이셔널 데이터 모델은 계층 모델들이며, IDMS와 같은 수많은 제품에 설치된 IBM's IMS system, 그리고 the Codasyl model (Network model)가 전형이다.

1) navigational database is a type of database in which records or objects are found primarily by following references from other objects. Navigational interfaces are usually procedural, though some modern systems like XPath can be considered

to be simultaneously navigational and declarative.

Navigational access is traditionally associated with the network model and hierarchical model of database interfaces, and some have even acquired set-oriented features. Navigational techniques use "pointers" and "paths" to navigate among data records (also known as "nodes"). This is in contrast to the relational model (implemented in relational databases), which strives to use "declarative" or logic programming techniques that ask the system forwhat to fetch instead of how to navigate to it.

2) CODASYL (often spelled Codasyl) is an acronym for "Conference on Data Systems Languages". This was a consortium formed in 1959 to guide the development of a standard programming language that could be used on many computers. This effort led to the development of COBOL and other standards.

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 is made up of ledger-style tables, each used for a different type of entity. It was not until the mid-1980s that computing hardware became powerful enough to allow relational systems (DBMSs plus applications) to be widely deployed. By the early 1990s, however, relational systems were dominant for all large-scale data processing applications, and they remain dominant today (2014) except in niche areas. The dominant database language is the standard SQL for the relational model, which has influenced database languages for other data models.

Codd에 의해 1970년에 처음으로 제안된 관계형 모델은 어플이 연결된 링크보다는 콘텐트에 의해 데이터를 탐색해야 한다는 모델이므로 기존의 전통적 모델과는 차별화한 것이다. 관계형모델은 원부형태의 테이블로 구성되어 있으며 각각의 테이블은 서로 다른 엔티티용으로 사용되었다. 1980년대 중반이 돼서야, 컴퓨팅 하드웨어가 관계형 시스템(DBMSs 플러스 어플즈)을 받아들일 수 있을 만큼 강력하게 됨으로써 폭넓게 사용되었다. 1990년대 초에 이르러서야, 그렇지만, 관계형 시스템은 모든 대규모 데이터 처리 어플즈에서 사용되었으며, 그것들은 2014년 현재에도 특정분야를 제외하고는 대단한 위세를 떨치고 있다. 이것의 뛰어난 데이터베이스 언어는 관계형 모델용인 표준 SQL이며 이것은 다른 데이터 모델용의 데이터베이스 언어에도 영향을 끼치고 있다.

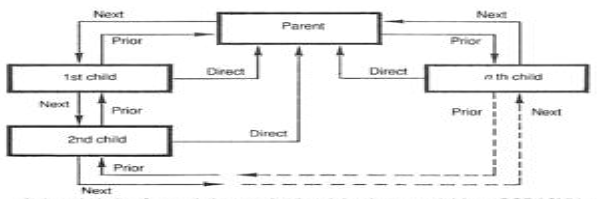
Object databases were invented in the 1980s to overcome the inconvenience of object-relational impedance mismatch, which led to the coining of the term "post-relational" but also development of hybrid object-relational databases. The next generation of post-relational databases in the 2000s became known as NoSQL databases, introducing fast key-value stores and document-oriented databases. A competing "next generation" known as NewSQL databases attempted new implementations that retain ed the relational/SQL model while aiming to match the high performance of NoSQL compared to commercially available relational DBMSs.

객체 데이터베이스는 1980년대에 object-relational impedance의 불일치라는 불편함을 개선하기 위하여 개발되었으며, 이것은 "후기 관계형"이라는 용어를 만드는데 일조하였을 뿐만 아니라 혼합형 객체-관계 데이터베이스의 개발을 불러 왔다. 2000년대에 차세대의 후기 관계형 데이터베이스는 신속한 key-value stores와 document-oriented databases를 도입함으로써 NoSQL 데이터베이스로 알려지게 되었다. NewSQL databases로 알려진 경쟁력 있는 "차세대"는 상업적으로 이용 가능한 관계형 DBMSs와 비교하여 NoSQL의 고성능을 따라잡는 것을 목표로 하지만 관계형/SQL 모델을 유지하는 새로운 실험을 시도하였다.

1) A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling and finer control over availability. NoSQL databases are often highly optimized key-value stores intended primarily for simple retrieval and appending operations, whereas an RDBMS is intended as a general purpose data store. There will thus be some operations where NoSQL is faster and some where an RDBMS is faster. NoSQL databases are finding significant and growing industry use in big data and real-time web applications. [1] NoSQL systems are also referred to as "Not only SQL" to emphasize that they may in fact allow SQL-like query languages to be used. In the context of the CAP theorem, NoSQL stores often compromise consistency in favor of availability and partition tolerance. Barriers to the greater adoption of NoSQL data stores in practice include: the lack of full ACID transaction support, the use of low-level query languages, the lack of standardized interfaces, and the huge investments already made in SQL by enterprises.

3.1 1960s Navigational DBMS

<Basic structure of navigational CODASYL database model>



A closed chain of records in a navigational database model (e.g. CODASYL), with ***next pointers***, ***prior pointers*** and ***direct pointers*** provided by keys in the various records.

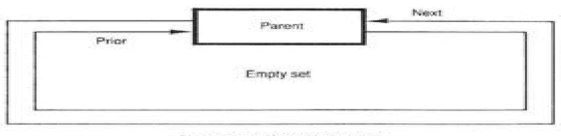


Illustration of an ***empty set***

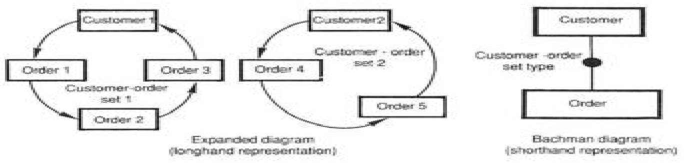


Illustration of a set type using a ***Bachman diagram***

The record set, basic structure of navigational (e.g. CODASYL) databse model. A set consists of one parent record (also called "the owner"), and n child records (also called members records)

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.

데이터베이스라는 단어의 도입은 1960년대 중반 이전부터 직접접근저장매체(디스크와 드럼)의 이용이 가능하면서 이루어졌다. 그 용어는 과거의 테이프 의존형 시스템과 대조적으로 사용되었으며 매일의 배치처리보다는 공유하면서 쌍방형의 사용이 가능하다는 것을 의미하게 되었다. The Oxford English dictionary에서는 특별한 기술적 의미로 “data-base”라는 용어를 처음으로 사용한 것이 the System Development Corporation of California에서 작성한 1962년도 보고서라고 말하고 있다.

As computers grew in speed and capability, a number of general-purpose database systems emerged; by the mid-1960s there were a number of such systems in commercial use. Interest in a standard began to grow, and Charles Bachman, author of one such product, the Integrated Data Store (IDS), founded within CODASYL, the group responsible for the creation and standardization of COBOL. In 1971 they delivered their standard, which generally became known as the "Codasyl approach", and soon a number of commercial products based on this approach were made available.

컴퓨터의 속도와 용량이 증가함으로써, 수많은 범용 데이터베이스 시스템이 등장하였다:

1960년대 중반까지, 수많은 이러한 시스템이 상업적 용도로 존재하였다. 표준에 대한 관심이 대두하였으며, the Integrated Data Store (IDS)과 같은 제품의 창시자인 Charles Bachman은 COBOL의 제작과 표준화에 책임을 가진 그룹인 CODASYL의 the "Database Task Group"을 설립하였다. 1971년에 그들은 자신들의 표준을 보급하였으며 일반적으로 이것은 Codasyl approach로 알려지게 되었고 곧 수많은 상업적 제품들이 이 어프로치를 근거로 하여 이용 가능하게 만들어졌다.

1) CODASYL (often spelled Codasyl) is an acronym for "Conference on Data Systems Languages". This was a consortium formed in 1959 to guide the development of a standard programming language that could be used on many computers. This effort led to the development of COBOL and other standards.

The Codasyl approach was based on the "manual" navigation of a linked data set which was formed into a large network. Records could be found either by use of a primary key (known as a CALC key, typically implemented by hashing), by navigating relationships (called sets) from one record to another, or by scanning all the records in sequential order. Later systems added B-Trees to provide alternate access paths. Many Codasyl databases also added a query language that was very straightforward. However, in the final tally, CODASYL was very complex and required significant training and effort to produce useful applications.

Codasyl approach는 하나의 커다란 네트워크 속에 형성되어 있는 링크된 데이터 세트를 수작업으로 항해하는 것을 기본으로 하고 있다. 레코드들은 으뜸키(known as a CALC key, typically implemented by hashing)를 사용하거나 한 레코드에서 다른 레코드로 그것들의 관계를 항해함으로써 또는 순차적으로 모든 레코드를 스캐닝 함으로써 찾을 수 있다. 그 후의 시스템들은 B-Tree를 추가하여 또 다른 대안적 접근 통로를 제공하였다. 많은 Codasyl 데이터베이스들 또한 쿼리 언어를 추가함으로써 매우 직선적이 되었다. 그렇지만, 최종 버전에서, CODASYL은 매우 복잡하였으며 유익한 어플즈를 생산하기 위해서는 많은 훈련과 노력을 요구하게 되었다.

1) a B-tree is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children.

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

IBM 또한 1968년에 자신들 만의 DBMS 시스템을 보유하였으며, 이것이 IMS이다. IMS는 System/360에서 사용하기 위하여 아폴로 프로그램용으로 작성된 발달된 소프트웨어였다. IMS는 일반적으로 Codasyl의 개념과 유사했지만, Codasyl의 네트워크 모델 대신에 자체의 데이터 항해용 모델을 위해 엄격한 계층적 모델을 사용하였다. 두 가지 개념 모두 나중에 데이터의 접근 방법에 따라 navigational databases로 알려지게 되었으며, Bachman의 1973 Turing Award 의 presentation은 The Programmer as Navigator였다. IMS는 계층적 데이터베이스로 분류되고 있으며, IDMS와 Cincom Systems' TOTAL 데이터베이스는 네트워크 데이터베이스로 분류되고 있다.

1) The network model is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or lattice.

3.2 1970s relational DBMS

Edgar Codd worked at IBM in San Jose, California, in one of their offshoot offices that was 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.

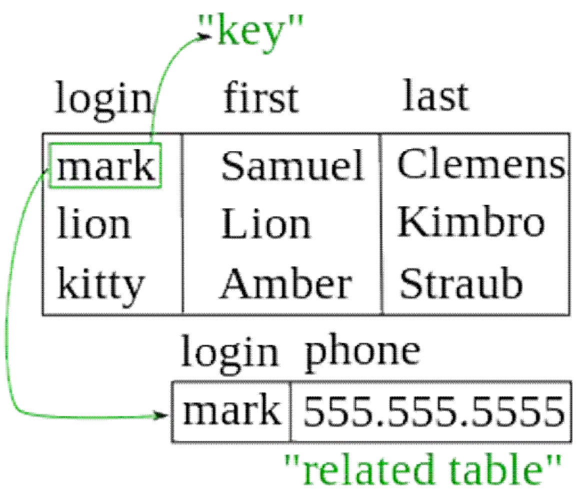
Edgar Codd는 IBM에서 근무할 때 하드 디스크 시스템의 개발에 처음으로 참여하였다. 그는 Codasyl approach의 항해모델에 대하여 불만족스러웠으며 특히 탐색 성능에 그러했다. 1970에, 그는 많은 논문을 써서 데이터베이스 구축의 새로운 방식의 틀을 마련했는데, 이것이 결과적으로 놀라운 A Relational Model of Data for Large Shared Data Banks의 토대가 되었다.

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 use a "table" of fixed-length records, with each table used for a different type of entity. A linked-list system would be very inefficient when storing "sparse" databases where some of the data for any one record could be left empty. The relational model solved this by splitting the data into a series of normalized tables (or relations), with optional elements being moved out of the main table to where they would take up room only if needed.

Data may be freely inserted, deleted and edited in these tables, with the DBMS doing whatever maintenance needed to present a table view to the application/user.

이 논문에서 그는 대용량의 데이터베이스에서 작업하고 저장하는 새로운 시스템을 설명하였다. Codasyl에서처럼 자유로운 형태의 레코드로 되어 있는 링크 리스트의 몇몇 종류에 저장되어 있는 레코드 대신에, Codd의 아이디어는 서로 다른 유형의 객체별로 각각의 테이블을 사용하는 고정장 레코드의 테이블을 사용하는 것이었다. 링크 리스트 시스템은 어떤 한 개의 레코드용의 데이터가 빈 공간으로 남아있게 되는 '성근' 데이터베이스를 저장할 때 매우 비효율적일 수 있다. 관계형 모델은 주 테이블로부터 필요하다면 공간을 차지할 수 있는 장소로 이동이 가능하도록 하는 선택 요소와 더불어 그 데이터를 일련의 정규화 테이블(또는 관계)로 쪼갬으로써 이것을 해결하였다. 데이터는 이러한 테이블 내에서 자유롭게 입력, 삭제, 그리고 편집될 수 있다. 따라서 DBMS는 어플/이용자에게 table view를 제공하는데 필요한 어떠한 유지관리도 가능하게 되었다.

In the relational model, related records are linked together with a "key“



The relational model also allowed the content of the database to evolve without constant rewriting of links and pointers. The relational part comes from entities referencing other entities in what is known as one-to-many relationship, like a traditional hierarchical model, and many-to-many relationship, like a navigational (network) model. Thus, a relational model can express both hierarchical and navigational models, as well as its native tabular model, allowing for pure or combined modeling in terms of these three models, as the application requires.

관계형 모델은 또한 데이터베이스의 콘텐트로 하여금 링크와 포인터에 대한 지속적인 재작업 없이도 진화하도록 한다. 관계형 부분은 전통적인 계층모델 같이 소위 일-대-다로, 그리고 항해 모델 같이 다-대-다로 알려진 다른 객체를 참조하는 객체들로부터 이루어진다. 그러므로 관계형 모델은 계층적 그리고 항해적 모델 둘 다 뿐만 아니라 그것의 고유한 태블러 모델도 표현할 수 있는데 왜냐하면 어플이 요구할 때 이러한 세 가지 모델과 관련해서 순수하거나 결합된 모델링이 가능하기 때문이다.

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

예를 들어, 데이터베이스 시스템의 일반적 용도는 이용자, 그들의 이름, 로그인 정보, 다양한 주소와 전화번호에 대한 정보를 추적하는 것이다. 항해적 어프로치에서, 모든 이러한 데이터는 하나의 단일 레코드에 자리 잡고 있을 것이며 사용되지 않는 아이템은 단지 그 데이터베이스 속에 자리 잡지 못할 것이다. 관계형 어프로치에서, 데이터는 이용자 테이블, 주소 테이블, 그리고 전화번호 테이블로 예를 들어 정규화 될 것이다. 단지 주소나 전화번호가 실재로 제공될 때만 레코드들이 이들 선택적 테이블 속에 만들어질 것이다.

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

정보를 함께 뒤로 링크하는 것이 이 시스템의 키이다. 관계형 모델에서, 어느 정도의 정보는 “키”로 이용되며, 이것은 특별한 레코드를 유일하게 정의하는데 사용된다. 이용자에 대한 정보가 수집될 때, 선택적 테이블에 저장된 정보는 이 키를 가지고 탐색함으로써 찾을 수 있다. 예를 들어, 이용자의 로그인 이름이 유일하다면, 그 이용자의 주소와 전화번호는 그것의 키로서 로그인 이름과 함께 레코드 될 것이다. 단일 콜렉션으로 다시 되돌아가도록 관련된 데이터를 이처럼 간단하게 “재-링킹”하는 것은 전통적인 컴퓨터 언어로서는 결코 디자인할 수 없는 것이다.

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

항해 어프로치가 레코드를 수집하기 위하여 loop하는 프로그램을 필요로 하는 것과 마찬가지고, 관계형 어프로치도 어떤 하나의 레코드에 대한 정보를 수집하기 위하여 loop를 필요로 한다. 필요로 하는 looping에 대한 Codd의 해결책은 집합-지향성 언어이며, 이것은 나중에 유비쿼터스 SQL을 탄생시킬 제안이었다. tuple calculus로 알려진 수학의 일부를 사용함으로써, 그는 그러한 시스템은 한 번의 운영만으로도 데이터의 세트를 발견하고 불러오는 간단한 시스템을 제공할 뿐만 아니라 정상적인 데이터베이스의 모든 운영(입력, 갱신 등)을 지원할 수 있다는 것을 보여주었다.

1) Since the calculus is a query language for relational databases we first have to define a relational database. The basic relational building block is the domain, or data type. A tuple is an ordered multiset of attributes, which are ordered pairs of domain and value; or just a row. A relvar (relation variable) is a set of ordered pairs of domain and name, which serves as the header for a relation. A relation is a set of tuples. Although these relational concepts are mathematically defined, those definitions map loosely to traditional database concepts. A table is an accepted visual representation of a relation; a tuple is similar to the concept of row.

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.

Codd의 논문은 버클리에서 두 사람, Eugene Wong 와 Micael Stonebraker에 의해 지지를 받았다. 이들은 이미 지리적 데이터베이스 프로젝트용으로 그리고 코드를 작성하기 위하여 학생 프로그래머용으로 할당되어 있는 기금을 사용하여 INGRES로 알려진 프로젝트를 시작하였다. 1973년에 시작된 INGRES는 1979년에 일반적으로 널리 사용하도록 하기 위한 첫 번째 시제품을 생산하였다. INGRES는 QUEL로 알려진 데이터 접근용 언어의 사용을 포함하여 많은 분야에서 System R과 비슷했다. 시간이 지나면서, INGRES는 새롭게 등장하는 SQL 표준이 되었다.

1) Ingres Database is a commercially supported, open-source SQL relational database management system intended to support large commercial and government applications.

2) IBM System R is a database system built as a research project at IBM's San Jose Research Laboratory beginning in 1974. System R was a seminal project: it was the first implementation of SQL, which has since become the standard relational data query language. It was also the first system to demonstrate that a relational database management system could provide good transaction processing performance.

3) QUEL is a relational database query language, based on tuple relational calculus, with some similarities to SQL. It was created as a part of the Ingres DBMS effort at University of California, Berkeley, based on Codd's earlier suggested but not implemented Data Sub-Language ALPHA.

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.

IBM은 자체적으로 관계형 모델인 PRTV와 생산품인 Business System 12를 실험하였으나 현재에는 중단되었다. Honeywell은 Multics용인 MRDS을 만들었으며 지금은 2개의 새로운 implementation(=technical specification)인 Alphora Dataphor과 Rel이 사용되고 있다. 관계형이라고 부르는 대부분의 다른 DBMS implementations는 실재로는 SQL DBMSs이다.

1) The Multics Relational Data Store, or MRDS for short, was the first commercial relational database management system. It was written PL/1 by Honeywell for the Multics operating system and first sold in June 1976. Unlike the SQL systems that emerged in the late 1970s and early 80's, MRDS used a command language only for basic data manipulation, equivalent to the SELECT or UPDATE statements in SQL.

2) Dataphor is an open-source truly-relational database management system (RDBMS) and its accompanying user interface technologies, which together are designed to provide highly declarative software application development. The Dataphor Server has its own storage engine or it can be a virtual, or federated, DBMS, meaning that it can utilize other database engines for storage.

3) Rel is an open source true relational database management system that implements a significant portion of Chris Date and Hugh Darwen's Tutorial D query language. Primarily intended for teaching purposes, Rel is written in the Java programming language.

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.

1970년에, 미시간 대학에서 D.L. Childs' Set-Theoretic Data model을 근거로 MICRO Information Management System의 개발을 시작하였다. Micro는 미국 노동부, 미국 환경보호기구, 그리고 알베르타대학, 미시간대학, 그리고 웨인 주립대학의 연구자들에 의해 대량의 데이터 세트를 관리하는데 사용되었다. 이것은 the Michigan Terminal System을 사용하는 IBM의 대형 컴퓨터에서 기동되었다.

3.3 Database machines and appliances

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 lower cost. Examples were IBM System/38, the early offering of Teradata, and the Britton Lee, Inc. database machine.

1970년대와 1980년대에 통합형 하드웨어와 소프트웨어를 갖춘 데이터베이스 시스템의 구축에 대한 시도가 있었다. 그것의 근저가 되는 철학은 그 같은 통합이 저비용으로 고성능을 제공할 수 있다는 것이다. IBM System/38, Teradata의 초기 버전, Britton Lee, Inc.의 database machine이 그 예들이다.

1) The System/38 had 48-bit addressing, which was unique for the time, and a novel integrated database system. The operating system of the System/38 was called CPF, for "Control Program Facility" CPF is not related to SSP, the operating system of the IBM System/34 and System/36.

2) Teradata Corporation is an American computer company that sells analytic data platforms, applications and related services. Its products are meant to

consolidate data from different sources and make the data available for analysis.

3) Britton Lee Inc. was a pioneering relational database company. Renamed ShareBase, it was acquired by Teradata in June, 1990.

Another approach to hardware support for database management was ICL's CAFSaccelerator, 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 for certain applications by some companies like Netezza and Oracle(Exadata).

데이터베이스 관리를 위한 하드웨어의 지원과 관련된 또 다른 시도는 ICL의 CAFS 추진기인데, 이것은 프로그램이 가능한 탐색성능을 갖춘 하드웨어 디스크 제어기이다. 결국, 이러한 노력들은 일반적으로 성공하지 못했는데, 그 이유는 전문화된 데이터베이스 기계들이 범용 컴퓨터의 급속한 발전을 따라갈 수 없었기 때문이다. 따라서 대부분의 데이터베이스 시스템은 오늘날 범용 컴퓨터 데이터 저장매체를 사용하는 범용 하드훼어에서 기동되는 소프트웨어 시스템들이다. 그렇지만 이러한 아이디어는 아직까지 Netezza와 Oracle(Exadata)과 같은 몇몇 회사에 의해 어떤 어플즈용으로 진행되고 있다.

1) International Computers Limited, or ICL, was a large British computer hardware, computer software and computer services company that operated from 1968 until 2002. It was formed through a merger of International Computers and Tabulators (ICT), English Electric Leo Marconi (EELM) and Elliott Automation in 1968. The company's most successful product line was the ICL 2900 Series range of mainframe computers.

2) The Content Addressable File Store (CAFS) was a hardware device developed by International Computers Limited (ICL) that provided a disk storage with built-in search capability. The motivation for the device was the discrepancy between the high speed at which a disk could deliver data, and the much lower speed at which a general-purpose processor could filter the data looking for records that matched a search condition.

3) Netezza (pronounced Ne-Tease-Ah) designs and markets high-performance data warehouse appliances and advanced analytics applications for uses including enterprise data warehousing, business intelligence, predictive analytics and business continuity planning.

4) Oracle Corporation is an American multinational computer technology corporation headquartered in Redwood City, California, United States. The company specializes in developing and marketing computer hardware systems and enterprise software products – particularly its own brands of database management systems. Oracle is the second-largest software maker by revenue, after Microsoft. The company also builds tools for database development and systems of middle-tier software, enterprise resource planning software (ERP), customer relationship management software (CRM) and supply chain management (SCM) software.

5) Oracle Exadata is a database appliance with support for both OLTP (transactional) and OLAP (analytical) database systems. It was initially designed in collaboration between Oracle Corporation and Hewlett Packard. Oracle designed the database, operating system (based on the Oracle Linux distribution), and storage software whereas HP designed the hardware for it. After Oracle's acquisition of Sun Microsystems, in 2010 Oracle announced the Exadata Version 2 with improved performance and Sun storage systems.

3.4 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 – 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 (DB2).

IBM은 1970년대 초에 System R처럼 Codd의 개념을 근거로 허술한 시제품 시스템에서 작업을 시작하였다. 첫 번째 버전은 1974/5년에 나왔으며 그 다음으로는 (선택 가능한) 레코드용의 모든 데이터가 하나의 커다란 “chunk(a fragment of information used in many multimedia formats)”에 저장될 필요가 없으므로 그 데이터를 세분할 수 있는 멀티-테이블 시스템을 시작하였다. 후속편의 멀티유저 버전은 1978과 1979년에 소비자에 의해 테스트 되었으며, 그 즈음에 표준화된 쿼리 언어인 SQL이 추가되었다. Codd의 아이디어는 Codasyl보다 우수하고 작업도 가능한 것이 밝혀졌으며, 그로 인하여 IBM은 SQL/DS로 알려진, 그리고 나중에 Database2(DB 2)로 알려진 System R의 완전한 버전을 개발하게 되었다.

1) IBM DB2 is a family of database server products developed by IBM. These products all support the relational model, but in recent years some products have

been extended to support object-relational features and non-relational structures, in particular XML.

Larry Ellison's Oracle started from a different chain, based on IBM's papers on System R, and beat IBM to market when the first version was released in 1978. 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).

Larry Ellison의 Oracle은 System R에 관한 IBM의 논문을 근거로 여러 가지 chain으로부터 출발하여 첫 번째 버전이 1978년에 출시되었을 때 시장에서 IBM을 물리쳤다. Stonebraker는 새로운 데이터베이스인 Postgres – 지금은 PostgreSQL - 를 개발하기 위하여 INGRES에서 얻은 경험을 적용하였다. PostgreSQL은 종종 global mission critical applications용으로 사용되고 있다. (.org와 .inf인 도메인 네임 등기부는 자신들의 중요한 데이터 저장장치로서 이것을 사용하며, 많은 대기업과 금융기관에서도 이것을 사용하고 있다)

1) PostgreSQL, often simply Postgres, is an open source object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance.

In Sweden, Codd's paper was also read and Mimer SQL was developed from the mid-1970s at Uppsala University. In 1984, this project was consolidated into an independent enterprise. In the early 1980s, Mimer introduced transaction handling for high robustness in applications, an idea that was subsequently implemented on most other DBMS.

스웨덴에서, Codd의 논문은 또한 읽혀졌으며, 웁살라 대학에서 1970년대 중반부터 Mimer SQL을 개발하였다. 1984년에 이 프로젝트는 독립기업에 통합되었다. 1980년대 초에, Miner는 어플즈에서 높은 robustness(튼튼, 강건)를 위하여 transactio handling을 소개하였다. 이 아이디어는 그 후에 다른 대부분의 DBMS에서 실행되었다.

1) Mimer SQL is an SQL-based relational database management system from the Swedish company Mimer Information Technology AB (formerly: Upright Database Technology AB), which has been developed and produced since the 1970s. The Mimer SQL database engine is available for Microsoft Windows, Mac OS X, Linux, Symbian OS, Unix, VxWorks and OpenVMS. Unlike other competing DBMSes, Mimer only implements optimistic concurrency control.

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 have become irrelevant.

또 다른 데이터 모델인 객체-관계 모델은 1976년에 나타났으며, 초기의 관계형 모델보다 더욱 친숙한 설명을 강조함으로써 데이터베이스 디자인용으로 인기를 끌었다. 나중에, 객체-관계 구조는 관계형 모델을 위한 하나의 데이터 모델링 구조로 탈바꿈하였으며, 이 둘 간의 상이함은 의미가 없게 되었다.

3.5 1980s desktop databases

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.

1980년대는 desktop computing의 시대를 예고하였다. 새로운 컴퓨터들이 Lotus 1,2,3와 같은 스프레드시트와 dBASE와 같은 데이터베이스 소프트웨어를 갖추고서 그들의 이용자가 더 많은 일을 할 수 있도록 하였다. dBASE는 가볍고 어떤 컴퓨터 이용자에게도 이해하기가 쉬웠다. dBASE 개발자인 C. Wayne Ratliff가 주장: “dBASE는 많은 지겨운 업무에서 이미 사용되었던 BASIC, C, FORTRAN, 그리고 COBOL과 같은 프로그램과는 다르다. 데이터 조작은 이용자에 의해서보다는 그 대신에 dBASE에 의해 이루어진다. 그러므로 이용자는 파일을 열고, 읽고, 닫고, 그리고 공간할당을 관리하는 것과 같은 지겨운 일로 혼란을 겪기보다는 그가 현재 하고 있는 일에 집중할 수 있다.” dBASE는 1980년대와 1990년 초기 최고의 판매 소프트웨어 타이틀 중의 하나였다.

1) dBASE database software was one of the first and in its day the most successful database management systems for microcomputers. [2] The dBASE system includes the core database engine, a query system, a forms engine, and a programming language that ties all of these components together. dBASE's underlying file format, the .dbf file, is widely used in applications needing a simple format to store structured data.

3.6 1980s object-oriented databases

The 1980s, 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 relations 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.

객체지향형 프로그래밍이 등장하면서 1980년대 내내 다양한 데이터베이스에서 데이터를 처리하는 방법에 대한 발전이 이루어졌다. 프로그래머와 디자이너들은 자신들의 데이터베이스에서 객체처럼 데이터를 취급하기 시작하였다. 만일 어떤 사람의 데이터가 한 데이터베이스에 있다면 그 사람의 속성, 즉 그들의 주소, 전화번호, 나이는 연고가 없는 데이터로 존재하는 대신에 그 사람에게 속한다는 것이다. 이것은 데이터간의 관계는 객체와 그것들의 속성에 대한 관계이지 개별 필드에 대한 관계는 아니라는 것을 의미한다. object-relational impedance mismatch란 프로그램화된 객체와 데이터베이스 테이블 간에 발생하는 번역의 불편함을 말한다. 객체 데이터베이스와 객체-관계 데이터베이스는 프로그래머들이 순수한 관계형 SQL에 대한 대안으로 사용할 수 있는 객체지향형 언어(때때로 SQL의 확장 형태로)를 제공함으로써 이런 문제를 해결하려고 한다. 프로그래밍 측면에서 보면, object-relational mappings (ORMs)으로 알려진 라이브러리들은 동일한 문제를 해결하려는 시도인 것이다.

1) Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

2) The object-relational impedance mismatch is a set of conceptual and technical difficulties that are often encountered when a relational database management system (RDBMS) is being used by a program written in an object-oriented programming language or style; particularly when objects or class definitions are mapped in a straightforward way to database tables or relational schema.

3) An object database (also object-oriented database management system) is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object databases are different from relational databases which are table-oriented. Object-relational databases are a hybrid of both approaches.

4) An object-relational database (ORD), or object-relational database management system (ORDBMS), is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, just as with pure relational systems, it supports extension of the data model with custom data-types and methods.

5) Object-relational mapping (ORM, O/RM, and O/R mapping) in computer software is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools.

3.7 2000s NoSQL and NewSQL databases[edit]

The next generation of post-relational databases in the 2000s became known as NoSQL databases, including fast key-value stores and document-oriented databases. XML databases are a type of structured document-oriented database that allows querying based on XML document attributes. 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.

2000년대에 후기 관계형 데이터베이스의 차세대는 NoSQL 데이터베이스로 알려졌으며, 이것에는 신속한 key-value stores와 document-oriented databases가 포함되어 있다. XML 데이터베이스는 한 종류의 정형화된 도큐멘트 지향형 데이터베이스이며, 이것은 XML 도큐멘트 속성들을 근거로 쿼리하는 것을 허용한다. NoSQL 데이터베이스는 종종 매우 빠르며, 고정된 테이블 스키마가 필요하지도 않고 비정규화 데이터를 저장함으로써 join 연산기능을 피하고, scare horizontally하게 디자인 되었다.

1) To scale horizontally (or scale out) means to add more nodes to a system, such as adding a new computer to a distributed software application. To scale vertically (or scale up) means to add resources to a single node in a system,

typically involving the addition of CPUs or memory to a single computer.

In recent years there was a high 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 maximum level of data consistency.

최근에 고성능의 partition tolerance를 갖춘 대규모의 분산 데이터베이스에 대한 요구가 높아지고 있지만, CAP 공론에 따라, 분산 시스템이 동시에 일관성, 이용가능성, 그리고 partition tolerance를 보장하는 것은 불가능하다. 분산 시스템은 이러한 보증들 중에서 어떤 두 가지는 동시에 만족시킬 수 있으나 세 가지 모두는 아니다. 그 같은 이유로 인하여 많은 NoSQL 데이터베이스들은 최대 수준의 데이터 일관성과 더불어 이용가능성과 partition tolerance 보증 둘 다를 제공하기 위하여 소위 eventual consistency를 사용하고 있다.

1) the CAP theorem, also known as Brewer's theorem, states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

* Consistency (all nodes see the same data at the same time)
* Availability (a guarantee that every request receives a response about whether it was successful or failed)
* Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system)

According to the theorem, a distributed system cannot satisfy all three of these guarantees at the same time.

2) Eventual consistency is a consistency model used in distributed computing that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

The most popular NoSQL systems include: MongoDB, Couchbase, Riak, Oracle NoSQL Database, memcached, Redis, CouchDB, Hazelcast, Apache Cassandra and HBase.

Note that all are open-source software products. A number of new relational databases continuing use of SQL but aiming for performance comparable to NoSQL are known as NewSQL.

대부분의 인기 있는 NoSQL 시스템에는 MongoDB, Couchbase, Riak, Oracle NoSQL

Database, memcached, Redis, CouchDB, Hazelcast, Apache Cassandra and HBase이 있으며, 모두가 오픈 소스 소프트웨어 제품들이다. SQL을 계속 사용하고 있지만 NoSQL과 견줄만한 성능을 목표로 하고 있는 수많은 새로운 관계형 데이터베이스들은 NewSQL로 알려져 있다.

1) Open-source software (OSS) is computer software with its source code made available and licensed with a license in which the copyright holder provides the rights to study, change and distribute the software to anyone and for any purpose. [1] Open-source software is very often developed in a public, collaborative manner.

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

4. Database 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 and 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, ACMPODS, VLDB, IEEE ICDE).

데이터베이스 기술은 1960년 이래로 학술기관이나 IBM Research와 같은 회사의 연구개발 부서에서 활발한 연구주제가 되었다. 연구활동에는 원형에 대한 이론과 개발이 포함되어 있다. 유명한 연구 주제들에는 모델, 핵심적인 트랜젝션 개념 그리고 관련된 동시발생 통제 기술, 쿼리 언어와 쿼리 최적화 방법, RAID 등이 포함되어 있다. 데이터베이스 연구 분야에는 ACM Transactions on Database Systems-TODS, Data and Knowledge Engineering -DKE와 같은 여러 전문 학술지와 ACM SIGMOD, ACMPODS, VLDB, IEEE ICDE의 연간 회의록이 있다.

1) RAID is a storage technology that combines multiple disk drive components into a logical unit for the purposes of data redundancy and performance improvement. Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the specific level of redundancy and performance

required.

The term "RAID" was first used by David Patterson, Garth A. Gibson, and Randy Katz at the University of California, Berkeley in 1987, standing for redundant array of inexpensive disks. Industry RAID manufacturers later tended to interpret the acronym as standing for redundant array of independent disks. RAID is now used as an umbrella term for computer data storage schemes that can divide and replicate data among multiple physical drives: RAID is an example of storage virtualization and the array can be accessed by the operating system as one single drive. The different schemes or architectures are named by the word RAID followed by a number (e.g. RAID 0, RAID 1). Each scheme provides a different balance between the key goals: reliability and availability, performance and capacity.

RAID levels greater than RAID 0 provide protection against unrecoverable (sector) read errors, as well as whole disk failure.

5. Database type examples

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.

데이터베이스를 분류하는 한 가지 방법은 그것들의 콘텐트의 종류, 예를 들어 서지, 문서-텍스트, 통계, 도는 멀티미디어 객체와 관련짓는 것이다. 또 다른 방법은 그것들의 응용분야, 예를 들어 회계, 음악, 연화, 금융, 제조, 또는 보험으로 하는 것이다. 세 번째 방법은 어떤 기술적인 관점, 예를 들어 데이터베이스 구조 또는 인터페이스 종류로 구분하는 것이다. 여기서는 다양한 종류의 데이터베이스를 특정하기 위하여 사용된 몇 가지의 형용사를 가지고 열거하기로 한다.

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. SAP HANAplatform is a very hot topic for in-memory database. By May 2012, HANA was able to run on servers with 100TB main memory powered by IBM. The co founder of the company claimed that the system was big enough to run the 8 largest SAP customers.

in-memory 데이터베이스는 기본적으로 주 메모리에 거주하고 있는 데이터베이스이지만 전형적으로 비휘발성 컴퓨터 데이터 저장매체에 의해 백업된다. 주 메모리 데이터베이스는 디스크 데이터베이스보다 훨씬 빠르며, 그래서 통신 네트워크 장비에서 응답시간이 중요할 때 종종 사용된다. SAP HANA 플랫폼은 in-memory 데이터베이스용으로 매우 뜨거운 주제이다. 2012년 5월까지, HANA는 IBM에서 공급한 100TB 주 메모리를 갖춘 서버에서 기동할 수 있었다. 그 회사의 공동 설립자가 주장하길 그 시스템은 8명의 가장 커다란 SAP 소비자가 운영할 수 있을 만큼 매우 충분하다고 하였다.

1) SAP HANA, short for 'High Performance Analytic Appliance' is an in-memory, column-oriented,

relational database management system developed and marketed by SAP AG.

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.

액티브 데이터베이스에는 그 데이터베이스의 안팎 조건에 응답할 수 있는 event-driven architecture가 포함된다. 잠재적 용도들로는 보안 감시, 경고, 통계, 수집과 인증이 포함된다. 많은 데이터베이스가 database triggers의 형태로 액티브 데이터베이스 특징을 제공하고 있다.

1) Event-driven architecture (EDA) is a software architecture pattern promoting the production, detection, consumption of, and reaction to events. An event can be defined as "a significant change in state". [1] For example, when a consumer purchases a car, the car's state changes from "for sale" to "sold". A car dealer's system architecture may treat this state change as an event whose occurrence can be made known to other applications within the architecture.

2) A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for maintaining the integrity of the information on the database. For example, when a new record (representing a new worker) is added to the employees table, new records should also be created in the tables of the taxes, vacations and salaries.

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 utilized by (application's) end-users through a web browser and Open APIs.

클라우드 데이터베이스는 클라우드 기술에 의존한다. 데이터베이스와 그것의 대부분의 DBMS는 둘 다 멀리 떨어져 “클라우드 속에” 존재하고 있다. 반면에 그것의 어플즈는 프로그래머에 의해 개발되어 나중에 웹 브라우저나 Open APIs를 통해 최종 이용자에 의해 유지 관리되고 활용된다.

1) A cloud database is a database that typically runs on a cloud computing platform, such as Amazon EC2, GoGrid, Salesforce and Rackspace. 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.

2) Cloud computing is a phrase used to describe a variety of computing concepts that involve a large number of computers connected through a real-time communication network such as the Internet. In science, cloud computing is a synonym for distributed computing over a network, and means the ability to run a program or application on many connected computers at the same time. The phrase also more commonly refers to network-based services, which appear to be provided by real server hardware, and are in fact served up by virtual hardware, simulated by software running on one or more real machines. Such virtual servers do not physically exist and can therefore be moved around and scaled up or down on the fly without affecting the end user, somewhat like a cloud.

In common usage, the term "the cloud" is essentially a metaphor for the Internet. Marketers have further popularized the phrase "in the cloud" to refer to software, platforms and infrastructure that are sold "as a service", i.e. remotely through the Internet. Typically, the seller has actual energy-consuming servers which host products and services from a remote location, so end-users don't have to; they can simply log on to the network without installing anything. The major models of cloud computing service are known as Software as a Service, Platform as a Service, and Infrastructure as a Service. These cloud services may be offered in a Public, Private or Hybrid network. Google, Inc. is one of the most well-known cloud vendors.

3) Open API (often referred to as OpenAPI new technology) is a word used to describe sets of technologies that enable websites to interact with each other by using REST, SOAP, JavaScript and other web technologies. While its possibilities aren't limited to web-based applications, it's becoming an increasing trend in

so-called Web 2.0 applications. The term API stands for Application Programming Interface.

The term "Open API" has been recently in use by recent trends in social media and Web 2.0. It is currently a heavily sought after solution to interconnect websites in a more fluid user-friendly manner. Open API also applies to collaborative services environments where managed service providers can also outsource specific services to other providers via systems integration. For example, companies like Level Platforms provide an open API to adapt to any business offering within the managed service environment. With the advent of the Facebook Platform, launched June 1st 2007, Facebook incorporated an open API into its business model.

OpenSocial is currently being developed by Google in conjunction with MySpace and other social networks including Bebo.com, Engage.com, Friendster, hi5, Hyves, imeem, LinkedIn, Ning, Oracle, orkut, Plaxo, Salesforce.com, Six Apart, Tianji, Viadeo, and XING. The ultimate goal is for any social website to be able to implement the APIs and host third party social applications.

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 retrieving, analyzing, and mining data, transforming, loading and managing data so as to make them available for further use.

데이터 웨어하우스는 업무 데이터베이스에서 나온 데이터와 종종 시장조사회사와 같은 외부정보원에서 나온 데이터를 보관하고 있으며, 업무데이터에 접근하기 어려운 또 다른 최종 이용자와 관리자가 사용할 수 있는 데이터의 중심원이 되었다. 예를 들어, 판매 데이터는 매주 합산하여 수집하여 UPCs를 사용하여 내부 제품코드로 바꿀 수도 있다. 그렇게 함으로써 그것들은 ACNielsen 데이터와 비교할 수 있다. 데이터 보관에 대한 몇 가지 기본적이고 필수적인 구성요소에는 미래의 이용이 가능하도록 하기 위하여 데이터의 검색, 분석, 발굴, 변형, 탑재, 관리가 포함된다.

1) The Universal Product Code (UPC) is a barcode symbology (i.e., a specific type of barcode) that is widely used in the United States, Canada, the United Kingdom, Australia, New Zealand and in other countries for tracking trade items in stores. Its most common form, the UPC-A, consists of 12 numerical digits, which are uniquely assigned to each trade item.

2) The Nielsen Corporation, also known as ACNielsen or AC Nielsen is a global marketing research firm, with worldwide headquarters in New York City, United States of America. Regional headquarters for North America are located in the Chicago suburb of Schaumburg, Illinois. As of May 2010, it is part of The Nielsen Company. This company was founded in 1923 in Chicago, by Arthur C. Nielsen, Sr., in order to give marketers reliable and objective information on the impact of marketing and sales programs.

A deductive database combines logic programming with a relational database, for example by using the Datalog language.

연역 데이터베이스는 예를 들어 Datalog 언어를 사용함으로써 관계형 데이터베이스와 논리적 프로그래밍을 결합한 것이다.

1) A Deductive database is a database system that can make deductions (i.e., conclude additional facts) based on rules and facts stored in the (deductive) database. Datalog is the language typically used to specify facts, rules and queries in deductive databases. Deductive databases have grown out of the desire to combine logic programming with relational databases to construct systems that support a powerful formalism and are still fast and able to deal with very large datasets. Deductive databases are more expressive than relational databases but less expressive than logic programming systems. In recent years, deductive databases such as Datalog have found new application in data integration, information extraction, networking, program analysis, security, and cloud computing.

2) Datalog is a truly declarative logic programming language that syntactically is a subset of Prolog. It is often used as a query language for deductive databases: it is more expressive than SQL. In recent years, Datalog has found new application in data integration, information extraction, networking, program analysis, security, and cloud computing.

A distributed database is one in which both the data and the DBMS span multiple computers.

분산 데이터베이스는 데이터와 그것의 DBMS 둘 다를 복수의 컴퓨터에 분산시켜 놓은 것이다.

1) A distributed database is a database in which storage devices are not all attached to a common processing unit such as the CPU, controlled by a

distributed database management system (together sometimes called a distributed database system). It may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely-coupled sites that share no physical components.

System administrators can distribute collections of data (e.g. in a database) across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. Because they store data across multiple computers, distributed databases can improve performance at end-user worksites by allowing transactions to be processed on many machines, instead of being limited to one.
Two processes ensure that the distributed databases remain up-to-date and current: replication and duplication.

Replication involves using specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be complex and time-consuming depending on the size and number of the distributed databases. This process can also require a lot of time and computer resources.

Duplication, on the other hand, has less complexity. It basically identifies one database as a master and then duplicates that database. The duplication process is normally done at a set time after hours. This is to ensure that each distributed location has the same data. In the duplication process, users may change only the master database. This ensures that local data will not be overwritten.
Both replication and duplication can keep the data current in all distributive locations.

Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementation can and does depend on the needs of the business and the sensitivity/confidentiality of the data stored in the database, and hence the price the business is willing to spend on ensuring data security, consistency and integrity.

When discussing access to distributed databases, Microsoft favors the term distributed query, which it defines in protocol-specific manner as "[a]ny SELECT, INSERT, UPDATE, or DELETE statement that references tables and rowsets from one or more external OLE DB data sources". Oracle provides a more language-centric view in which distributed queries and distributed transactions form part of distributed SQL.

A document-oriented database is designed for storing, retrieving, and managing document-oriented, or semi structured data, information. Document-oriented databases are one of the main categories of NoSQL databases.

도큐멘트 지향형 데이터베이스는 도큐멘트 위주 또는 반-정형화 데이터인 정보를 저장, 검색, 관리하도록 디자인되었다. 도큐멘트 지향형 데이터베이스는 NoSQL 데이터베이스의 주요 부류 중의 하나이다.

1) The central concept of a document-oriented database is the notion of a Document. While each document-oriented database implementation differs on the details of this definition, in general, they all assume documents encapsulate and encode data (or information) in some standard formats or encodings. Encodings in use include XML, YAML, JSON, and BSON, as well as binary forms like PDF and Microsoft Office documents (MS Word, Excel, and so on). Documents inside a document-oriented database are similar, in some ways, to records or rows in relational databases, but they are less rigid. They are not required to adhere to a standard schema, nor will they have all the same sections, slots, parts, or keys.

2) The semi-structured model is a database model where there is no separation between the data and the schema, and the amount of structure used depends on the purpose. The advantages of this model are the following:

a) It can represent the information of some data sources that cannot be constrained by schema.

b) It provides a flexible format for data exchange between different types of databases.

c) It can be helpful to view structured data as semi-structured (for browsing purposes).

d) The schema can easily be changed.

e) The data transfer format may be portable.

The primary trade-off being made in using a semi-structured database model is that queries cannot be made as efficient as in a more constrained structure, such as in the relational model. Typically the records in a semi-structured database are stored with unique IDs that are referenced with pointers to their location on disk. This makes navigational or path-based queries quite efficient, but for doing searches over many records (as is typical in SQL), it is not as efficient because it has to seek around the disk following pointers. The Object Exchange

Model (OEM) is one standard to express semi-structured data, another way is XML.

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.

내장형 데이터베이스 시스템은 저장된 데이터에 접근을 요구하는 하나의 어플 시스템으로 조밀하게 통합된 DBMS이다. 그 DBMS는 어플의 최종 이용자에게는 안보이며 어떠한 지속적인 관리도 거의 필요하지 않다.

1) An embedded database system is actually a broad technology category that includes database systems with differing application programming interfaces (SQL as well as proprietary, native APIs); database architectures (client-server and in-process); storage modes (on-disk, in-memory and combined); database models (relational, object-oriented, entity-attribute-value model and network/CODASYL); and target markets.

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 exist to support such databases. Some of them are much simpler than full fledged DBMSs, with more elementary DBMS functionality.

엔드유저 데이터베이스는 개별 최종이용자에 의해 개발된 데이터로 구성된다. 이러한 것들의 예로는 documents, spreadsheets, presentations, multimedia, and other files의 컬렉션들이다. 여러 가지 제품이 이러한 데이터베이스를 지원하기 위하여 존재한다. 이것들 어떤 것은 더 많은 기본적인 DBMS 기능성을 갖춤으로써 full fledges(최종판) DBMSs보다 훨씬 단순하다.

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.

연합 데이터베이스 시스템은 여러 가지의 차이가 나는 데이터베이스들로 구성되어 있으며, 각자는 자신만을 DBMS를 가지고 있다. 이것은 분명하게 아마도 서로 다른 종류(이러한 경우에는 이질적 데이터베이스 시스템일 수도 있다)인 복수의 자치적 EBMSs를 통합하고 있는 연방 데이터베이스 관리 시스템(FDBMS)에 의해 단일 데이터베이스로 취급되며, 하나의 통합된 개념적 뷰를 제공한다.

Sometimes the term multi-database is used as a synonym to 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.

때때로 멀티-데이터베이스라는 용어는 비록 그것이 하나의 단일 어플에서 협력하는 다소 통합력이 떨어진(예를 들어, FDBMS와 관리된 통합 스키마가 없는) 집단의 데이터베이스를 언급하더라도 연합 데이터베이스와 동의어로 사용된다. 이런 경우에 있어서 전형적으로 미들웨어는 분배용으로 사용되며, 이것에는 전형적으로 참가하고 있는 데이터베이스들 간에 distributed (golobal) transaction이 가능하도록, 예를 들어 the two-phase commit protocl과 같은 atomic commit protocol(ACP)가 포함된다.

1) Middleware in the context of distributed applications is software that provides services beyond those provided by the operating system to enable the various components of a distributed system to communicate and manage data. Middleware supports and simplifies complex distributed applications. It includes web servers, application servers, messaging and similar tools that support application development and delivery. Middleware is especially integral to modern information technology based on XML, SOAP, Web services, and service-oriented architecture.

2) the two-phase commit protocol (2PC) is a type of atomic commitment protocol (ACP). It is a distributed algorithm that coordinates all the processes that participate in a distributed atomic transaction on whether to commit or abort (roll back) the transaction (it is a specialized type of consensus protocol). The protocol achieves its goal even in many cases of temporary system failure (involving either process, network node, communication, etc. failures), and is thus widely utilized.

Middleware often enables interoperability between applications that run on different operating systems, by supplying services so the application can exchange data in a standards-based way. Middleware sits "in the middle" between application software that may be working on different operating systems. It is similar to the middle layer of a three-tier single system architecture, except that it is stretched across multiple systems or applications. Examples include EAI software, telecommunications software, transaction monitors, and messaging-and-queueing software.

3) A distributed transaction is an operations bundle, in which two or more

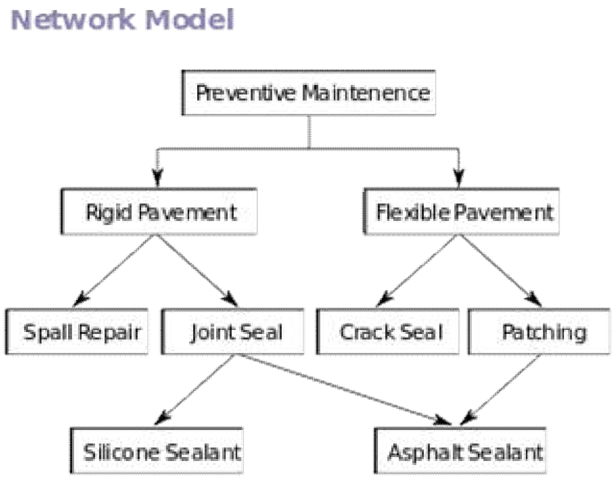
network hosts are involved. Usually, hosts provide transactional resources, while the transaction manager is responsible for creating and managing a global transaction that encompasses all operations against such resources. Distributed transactions, as any other transactions, must have all four ACID (atomicity, consistency, isolation, durability) properties, where atomicity guarantees all-or-nothing outcomes for the unit of work (operations bundle).

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.

그래픽 데이터베이스는 NoSQL 데이터베이스의 일종이며 정보를 저장하고 표현하기 위하여 nodes, edges, properties를 갖는 그래픽 구조를 사용한다. 어떠한 그래픽도 저장할 수 있는 일반적인 그래픽 데이터베이스는 triplestores와 network databases가 같은 전문 그래픽 데이터베이스와는 차이가 난다.

1) A triplestore is a purpose-built database for the storage and retrieval of triples,[1] a triple being a data entity composed of subject-predicate-object, like "Bob is 35" or "Bob knows Fred". Much like a relational database, one stores information in a triplestore and retrieves it via a query language. Unlike a relational database, a triplestore is optimized for the storage and retrieval of triples. In addition to queries, triples can usually be imported/exported using Resource Description Framework (RDF) and other formats.

2) The network model is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or lattice.



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.

하이퍼텍스트 또는 하이퍼미디어 데이터베이스에서, 객체를 표현하는 어떤 단어나 텍스트, 예를 들어 또 다른 텍스트, 그림, 필름은 다른 객체에 하이퍼링크 될 수 있다. 하이퍼텍스트 데이터베이스는 특히 대량의 이질적 정보를 조직하는데 유용하다. 예를 들어, 그것들은 이용자가 편리하고 텍스트 주위로 점프할 수 있는 온라인 백과사전을 만드는데 유용할 것이다. WWW는 그러므로 대규모의 분산 하이퍼텍스트 데이터베이스이다.

A knowledge base (abbreviated KB, kb) 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.

지식 베이스는 지식의 전산화된 수집, 조직, 검색을 위한 수단을 제공하는 지식관리를 위한 특별한 종류의 데이터베이스이다. 또한 해결책과 관련된 경험을 갖고 문제를 제시하는 데이터의 집단이다.

1) A knowledge base (KB) is a technology used to store complex structured and unstructured information used by a computer system. The initial use of the term was in connection with expert systems which were the first knowledge-based systems. The original use of the term knowledge-base was to describe one of the two sub-systems of a knowledge-based system. A knowledge-based system consists

of a knowledge-base that represents facts about the world and an inference engine that can reason about those facts and use rules and other forms of logic to deduce new facts or highlight inconsistencies.

2) an ontology formally represents knowledge as a set of concepts within a domain, using a shared vocabulary to denote the types, properties and interrelationships of those concepts.

3) Knowledge management (KM) is the process of capturing, developing, sharing, and effectively using organisational knowledge. It refers to a multi-disciplined approach to achieving organisational objectives by making the best use of knowledge.

A mobile database can be carried on or synchronized from a mobile computing device.

모바일 데이터베이스는 모바일 컴퓨팅 기기에서 운영되며 동기화할 수 있다.

1) mobile Devices database Management commonly called as Mobile Database' is either a stationary database that can be connected to by a mobile computing device - such as smart phones or PDAs - over a mobile network, or a database which is actually carried by the mobile device. This could be a list of contacts, price information, distance travelled, or any other information.

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' 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.

운영 데이터베이스는 조직의 운영에 대한 상세한 데이터를 저장한다. 이것들은 전형적으로 트랜젝션을 사용하여 비교적 많은 양의 갱신정보를 처리한다. 이것의 예로는 사업 고객에 대한 접촉, 신용, 그리고 인적 정보를 기록하고 있는 고객용 데이터베이스, 월급, 이윤, 종업원에 대한 기술 데이터, 제품 구성요소에 대한 상세 정보를 기록하고 있는 기업 자원 기획 시스템에 대한 정보를 소장하고 있는 인사 데이터베이스, 그리고 조직의 자금, 회계, 금융거래를 기록해 놓은 재정 데이터베이스가 있다.

A parallel database seeks to improve performance through parallelization for tasks such as loading data, building indexes and evaluating queries.

병행 데이터베이스는 데이터를 올리고, 색인을 만들고 쿼리를 평가하는 것과 같은 업무용으로 병렬처리를 통해 성능을 개선시키려고 한다.

1) Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel. Centralized and client-server database systems are not powerful enough to handle such applications. In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.

2) Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").

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

근간이 되는 하드웨어에 의해 만들어지는 주요한 병행 DBMS 구조는 다음과 같다:

Shared memory architecture, where multiple processors share the main memory space, as well as other data storage.

메모리를 공유하는 구조: 복수의 프로세서가 주 메모리 공간뿐만 아니라 기타 데이터 저장 공간도 공유한다.

Shared disk architecture, where each processing unit (typically consisting of multiple processors) has its own main memory, but all units share the other storage.

디스크를 공유하는 구조: 전형적으로 복수의 프로세서들로 이루어진 각각의 프로세싱 유니트가 자신만의 주 메모리를 갖고 있지만, 모든 유니트들이 다른 저장 공간을 공유한다.

Shared nothing architecture, where each processing unit has its own main memory and other storage.

아무것도 공유하지 않는 구조: 각각의 프로세싱 유니트가 자신만의 주 메모리와 다른 저장 공간을 갖고 있다.

Probabilistic databases employ fuzzy logic to draw inferences from imprecise data.

확률 데이터베이스: 부정확한 데이터로부터 추론을 이끌어내는 fuzzy logic을 사용한다.

1) Fuzzy logic is a form of many-valued logic; it deals with reasoning that is approximate rather than fixed and exact. Compared to traditional binary sets (where variables may take on true or false values) fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false.

<Example>

Hard science with IF-THEN rules

Fuzzy set theory defines fuzzy operators on fuzzy sets. The problem in applying this is that the appropriate fuzzy operator may not be known. For this reason, fuzzy logic usually uses IF-THEN rules, or constructs that are equivalent, such as fuzzy associative matrices.

Rules are usually expressed in the form:

IF variable IS property THEN action

For example, a simple temperature regulator that uses a fan might look like this:

IF temperature IS very cold THEN stop fan
IF temperature IS cold THEN turn down fan
IF temperature IS normal THEN maintain level
IF temperature IS hot THEN speed up fan

There is no "ELSE" - all of the rules are evaluated, because the temperature might be "cold" and "normal" at the same time to different degrees.

The AND, OR, and NOT operators of boolean logic exist in fuzzy logic, usually defined as the minimum, maximum, and complement; when they are defined this way, they are called the Zadeh operators. So for the fuzzy variables x and y:

NOT x = (1 - truth(x))
x AND y = minimum(truth(x), truth(y))
x OR y = maximum(truth(x), truth(y))

There are also other operators, more linguistic in nature, called hedges that can be applied. These are generally adverbs such as "very", or "somewhat", which modify the meaning of a set using a mathematical formula.

Real-time databases process transactions fast enough for the result to come back and be acted on right away.

실시간 데이터베이스: 곧바로 실행하거나 회수되어야 하는 결과에 충분하고도 신속하게 교신을 처리한다.

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?".

공간 데이터베이스: 다차원적인 모습으로 데이터를 저장할 수 있다. 그런 데이터에 대한 쿼리에는 "이 지역에서 가장 가까운 호텔은?"과 같은 위치기반 쿼리가 포함된다.

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.

일시적 데이터베이스: 제한된 시간 요소를 가지고 있다. 예를 들어, 임시 데이터 모델과 SQL의 임시 버전이 여기에 속한다. 좀 더 특별하게 말해서, 임시적 요소에는 대체로 유효시간과 거래시간이 포함된다.

A terminology-oriented database builds upon an object-oriented database, often customized for a specific field.
전문용어 데이터베이스: 객체지향형 데이터베이스에서 만들어지며, 종종 특정한 분야로 제한된다.

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.

비정형화 데이터 데이터베이스: 관리 및 보호가 가능한 방법으로 일반 데이터베이스에서는 자연스럽게 그리고 편리하게 다룰 수 없는 다양한 객체들을 저장하기 위한 목적을 갖고 있다. 여기에는 email messages, documents, journals, multimedia objects 등이 포함될 수

있다. 이 이름이 오해를 불러일으킬 수 있는데 왜냐하면 어떤 객체들은 매우 정형화되어있기 때문이다. 그렇지만, 모든 잠재적 객체 집단이 미리 설정된 정형화된 틀에 맞지는 않는다. 대부분의 기존의 DBMSs는 현재 다양한 방법으로 비정형화된 데이터를 지원하고 있으며, 새로운 전용 DBMSs도 출현하고 있다.

6. 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 organisation, 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.

데이터베이스 디자이너의 첫 번째 임무는 데이터베이스 속에 저장된 정보의 구조를 다룰 개념적 데이터 모델을 만드는 것이다. 이것을 하기 위한 일반적인 방법은 드로잉 도구를 사용하여 객체-관계 모델을 개발하는 것이다. 또 다른 인기 있는 방법은 the Unified Modeling Language 이다. 성공적인 데이터 모델은 모델화하려는 외부 세계의 잠재적 상태를 정확하게 반영하여야 할 것이다: 예를 들어, 만일 사람들이 하나이상의 전화번호를 가질 수 있다면, 이러한 정보를 수집할 수 있어야 할 것이다. 훌륭한 개념적 데이터 모델을 디자인하는 것은 어플 도메인에 대한 많은 이해를 필요로 한다; 전형적으로 여기에는 조직을 위해 관심의 대상이 되는 일에 대하여 상세하게 질문하는 것 – 예를 들어, “소비자가 또한 공급자가 될 수 있는가? ” 또는 “만일 하나의 제품이 두 개의 서로 다른 포장상태로 팔린다면, 이것들은 동일한 제품인가 또는 서로 다른 제품인가?” 또는 “비행기가 뉴욕에서 두바이를 거쳐 두바이까지 간다면, 그것은 한 번의 비행인가 또는 2번 또는 3번의 비행인가?”와 같은 질문이 포함된다. 이러한 질문들에 대한 해답들은 객체들(customers, products, flights, flight segments)과 그것들의 관계와 속성용으로 사용된 전문용어의 정의를 기술한다.

1) Unified Modeling Language (UML) is a standardized (ISO/IEC 19501:2005), general-purpose modeling language in the field of software engineering. The Unified Modeling Language includes a set of graphic notation techniques to create

visual models of object-oriented software-intensive systems.

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.

때때로 개념적 데이터 모델을 생산하는 데에는 사업 프로세서, 또는 기관의 업무흐름도의 분석에서 발생한 입력요소들이 포함된다. 이것은 데이터베이스에 필요한 정보가 무엇인지 그리고 보내야 할 것이 무엇인지를 파악하는데 도움을 줄 수 있다. 예를 들어, 이것은 데이터베이스가 역사적 데이터뿐 만 아니라 현재의 데이터를 보관하는 것이 필요한지에 대한 결정을 내릴 때 도움을 얻을 수 있다.

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 modelling notation used to express that design.)

이용자가 받아들이는 개념적 데이터 모델을 생산한 다음에, 그 다음 단계는 이것을 데이터베이스에서 적절한 데이터 구조로 실행하는 스키마로 변환시키는 것이다. 이 과정은 종종 논리적 데이터베이스 디자인이라고 부르며, 그 결과는 스키마의 형태로 표현된 논리적 데이터 모델이다. 개념적 데이터 모델이 적어도 이론적으로는 선택된 데이터베이스 테크놀로지와 상관없다 하더라도, 논리적 데이터 모델은 선택된 DBMS에 의해 지원을 받는 특별한 데이터베이스 모델과 관련해서 표현될 것이다(data model과 database model이란 용어들은 종종 호환적으로 사용되지만, 이 글에서 우리는 특별한 데이터베이스의 디자인을 위해서는 data model을 그리고 그 같은 디자인을 표현하기 위하여 사용되는 모델링 주석을 위해서는 database model을 사용한다.)

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.

일반용 데이터베이스용으로 가장 인기 있는 데이터베이스 모델은 관계형 모델이며, 더 정확하게 말해서 SQL 언어로 표현된 관계형 모델이다. 이러한 모델을 사용하여 논리적 데이터베이스 디자인을 제작하는 과정에서는 정규화로 알려진 방법론적 접근방식을 사용한다. 정규화의 목표는 각각의 기본적인 “사실”이 추가, 갱신, 그리고 삭제가 자동적으로 일관성 있게 이루어지도록 단지 한 장소에만 기록되도록 확실하게 하는 것이다.

1) Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database using the defined relationships.

Edgar F. Codd, the inventor of the relational model, introduced the concept of normalization and what we now know as the First Normal Form (1NF) in 1970. Codd went on to define the Second Normal Form (2NF) and Third Normal Form (3NF) in 1971, and Codd and Raymond F. Boyce defined the Boyce-Codd Normal Form (BCNF) in 1974. Informally, a relational database table is often described as "normalized" if it is in the Third Normal Form. Most 3NF tables are free of insertion, update, and deletion anomalies.

A standard piece of database design guidance is that the designer should first create a fully normalized design; then selective denormalization can be performed for performance reasons.[

The final stage of database design is to make the decisions that affect performance, scalability, recovery, security, and the like. This is often called physical database design. 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. 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.

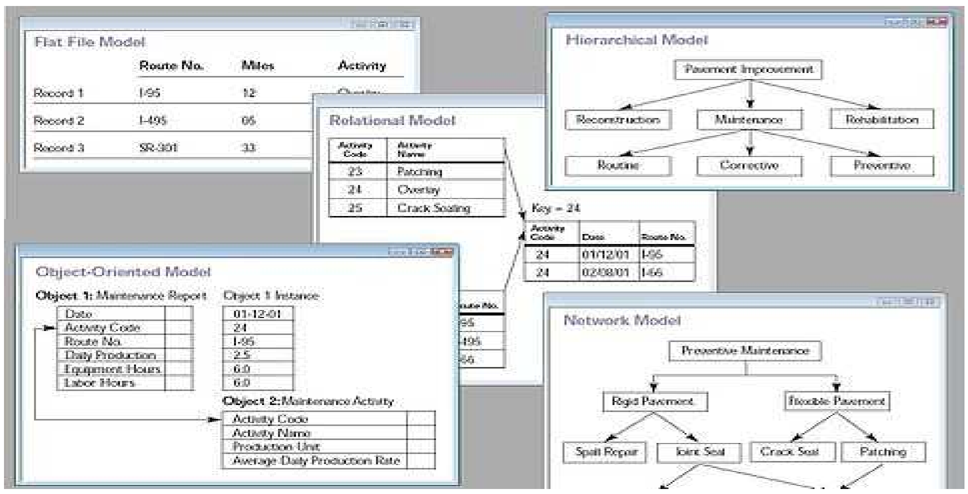
데이터베이스 디자인의 최종 단계는 성능, scalability, 회복력, 안전성 등에 영향을 끼치는 사안들에 대하여 결정하는 것이다. 이것은 종종 물리적 데이터베이스 디자인이라고 부른다. 이 단계 동안 중요한 목표는 데이터 독립성이다. 이것은 최적의 성능을 목표로 이루어진 의사결정이 최종 이용자와 어플에서는 시각화되지 않아야 한다는 것을 의미한다. 물리적 디자인은 주로 성능 요구서에서 필요로 하며 예상되는 업무량과 접근 패턴에 대한 충분한 지식, 그리고 선택된 DBMS에서 제공되는 특징에 대한 깊은 이해를 필요로 한다.

1) Scalability, as a property of systems, is generally difficult to define[2] and in any particular case it is necessary to define the specific requirements for scalability on those dimensions that are deemed important. It is a highly significant issue in electronics systems, databases, routers, and networking. A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a scalable system.

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.

또 다른 물리적 데이터베이스 디자인의 요소는 보안성이다. 이것에는 데이터베이스 객체에 대한 접근 통제를 정의하는 것뿐만 아니라 데이터 그 자체에 대한 보안 수준과 방법을 정의하는 것이 포함된다.

6.1. Database models



Collage of five types of database 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.

데이터베이스 모델은 데이터 모델의 한 종류이며 데이터베이스의 논리적 구조를 결정하고

근본적으로 어떠한 방식으로 데이터가 저장, 조직, 취급되는지를 결정한다. 가장 인기 있는 데이터베이스 모델은 테이블을 근거로 하는 형식의 관계형 모델(또는 관계형과 유사한 SQL 모델) 이다.

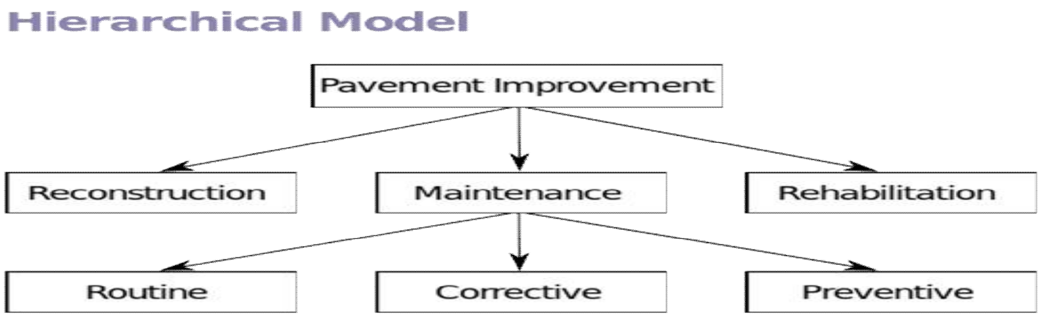
Common logical data models for databases include:

데이터베이스용의 일반적인 논리적 데이터 모델에는 다음과 같은 것이 포함된다:

Hierarchical database model

A hierarchical database model is a data model in which the data is organized into a tree-like structure. The structure allows representing information using parent/child relationships: each parent can have many children, but each child has only one parent (also known as a 1-to-many relationship). All attributes of a specific record are listed under an entity type.

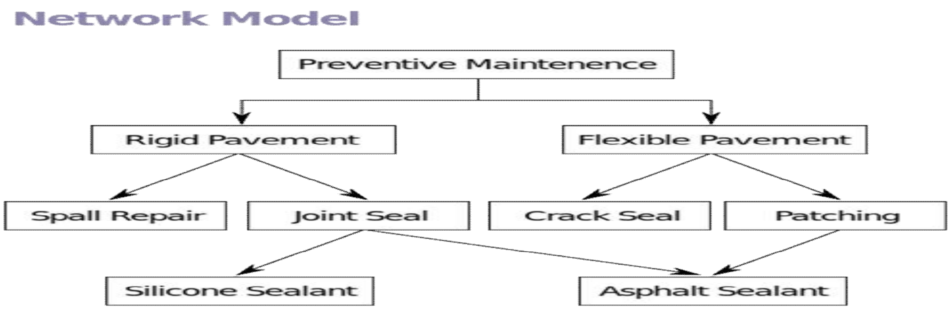
계층형 데이터베이스 모델은 데이터가 나무와 같은 구조로 조직된 데이터 모델이다. 이 구조에서는 부모/자식 관계를 사용하여 정보를 표현한다: 각 부모는 많은 자식을 가질 수 있으나 각 어린이는 단지 하나의 부모만을 갖는다(일-대-다의 관계로 알려져 있다). 특정한 한 레코드의 모든 속성은 하나의 객체 유형 아래에 열거된다.



Network model

The network model is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or lattice.

네트워크 모델은 객체와 그것들의 관계를 표현하는데 있어서 유연한 방법으로 인식된 데이터베이스 모델이다. 이것의 뛰어난 특징은 그 스키마(객체 유형은 nodes이고 관계유형은 아크인 그래프라고 여겨지는)가 계층이나 격자 형태로 되는 것을 제한하지 않는다.

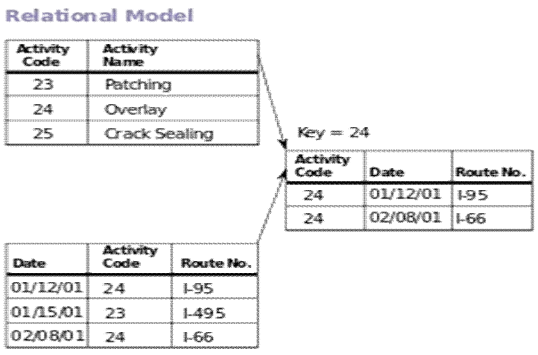


Relational model

The relational model for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar F. Codd. In the relational model of a database, all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.

데이터베이스 관리를 위한 관계형 모델은 Edgar F. Codd에 의해 1969년에 가장 먼저 제안되어 공식화된 first-order predicate logic을 근거로 하는 데이터베이스 모델이다. 데이터베이스 관계형 모델에서, 모든 데이터는 관계들을 집단화한 tuples로 표현된다.

관계형 모델로 조직된 데이터베이스는 관계형 데이터베이스이다.

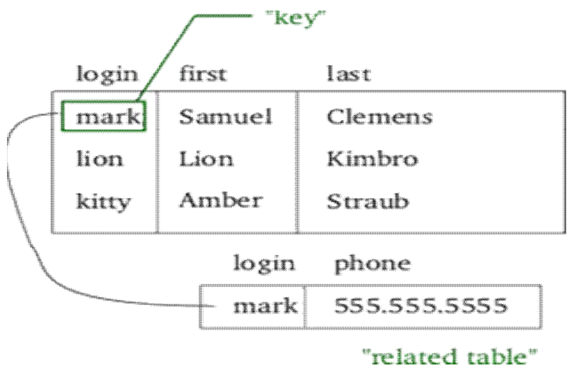


Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66

Diagram of an example database according to the Relational model.



In the relational model, related records are linked together with a "key"

The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

관계형 모델의 목적은 데이터와 쿼리를 특정화하기 위한 선언적 방법을 제공하는 것이다: 이용자가 직접적으로 데이터베이스에 들어있는 정보가 무엇이고 그들이 그것에서 원하는 정보가 무엇인지를 말하며, 그 데이터베이스 관리 시스템 소프트웨어는 쿼리에 답하도록 데이터를 저장하고 있는 데이터 구조와 검색절차의 기술을 관리한다.

Most relational databases use the SQL data definition and query language; these systems implement what can be regarded as an engineering approximation to the relational model. A table in an SQL database schema corresponds to a predicate variable; the contents of a table to a relation; key constraints, other constraints, and SQL queries correspond to predicates. However, SQL databases, including DB2, deviate from the relational model in many details, and Codd fiercely argued against deviations that compromise the original principles.

대부분의 관계형 데이터베이스는 SQL 데이터 정의와 쿼리 언어를 사용한다: 이러한 시스템들은 관계형 모델에서 공학적으로 유사한 것으로 간주된 것을 실행시킨다. SQL 데이터베이스 스키마에 있는 테이블은 서술적 변수(속성); 한 테이블의 콘텐트는 관계로; 키 제한조건, 기타 제한조건에 해당하며, SQL 쿼리는 서술(속성)에 해당한다. 그렇지만 SQL 데이터베이스, DB2를 포함하여 많은 부분에서 관계형 모델로부터 파생되었다. 그리고 Codd는 원래의 원칙과 타협하는 변종에 대하여 강력하게 비난하였다.

Entity-relationship model

An ER model is an abstract way of describing a database. In the case of a relational database, which stores data in tables, some of the data in these tables point to data in other tables - for instance, your entry in the database could point to several entries for each of the phone numbers that are yours. The ER model would say that you are an entity, and each phone number is an entity, and the relationship between you and the phone numbers is 'has a phone number'. Diagrams created to design these entities and relationships are called entity-relationship diagrams or ER diagrams.

ER 모델은 데이터베이스를 설명하는 추상적 방법이다. 테이블에 데이터가 저장되는 관계형 데이터베이스의 경우에, 이들 테이블에 있는 어떤 데이터는 다른 테이블에 있는 데이터를 지적한다. - 예를 들어, 데이터베이스에 있는 여러분의 입력사항은 여러분의 것인 각각의 전화번호용 복수의 기입사항을 지적한다. ER 모델은 여러분은 하나의 객체이고 각 전화번호도 하나의 객체이며 여러분과 전화번호와의 관계는 'has a phone number'라고 말하고 있다. 이러한 객체들과 관계들을 디자인하도록 만들어진 다이어그램은 객체-관계 다이어그램 또는 ER 다이어그램이라고 부른다.

Using the three schema approach to software engineering, there are three levels of ER models that may be developed.

소프트 공학용으로 3가지의 스키마 방법을 사용함으로써, 3가지 차원의 ER 모델을 개발할 수 있다.

Conceptual data model(개념적 데이터 모델)
This is the highest level ER model in that it contains the least granular detail but establishes the overall scope of what is to be included within the model set. The conceptual ER model normally defines master reference data entities that are commonly used by the organization. Developing an enterprise-wide conceptual ER model is useful to support documenting the data architecture for an organization.

이것은 가장 높은 수준의 ER 모델이며 그 속에는 최소한의 granular detail을 포함하고 있지만, 그 모델 세트에 포함되어야 하는 모든 대상에 대한 범위를 확립한다. 개념적 ER 모델은 보통은 그 기관에서 공동적으로 사용하는 master reference data 객체를 정의한다. 기업용 개념적 ER 모델을 개발하는 것은 그 기관의 데이터 구조를 기록화 하는 것을 지원하는데 유용하다.

1) Master data is also called Master reference data. This is to avoid confusion with the usage of the term Master data for original data, like an original recording (see also: Master Tape). Master data is nothing but unique data, i.e., there are no duplicate values.

A conceptual ER model may be used as the foundation for one or more logical data models (see below). The purpose of the conceptual ER model is then to establish structural metadata commonality for the master data entities between the set of logical ER models. The conceptual data model may be used to form commonality relationships between ER models as a basis for data model integration.

개념적 ER 모델은 하나 이상의 논리적 데이터 모델의 기초로서 사용될 수 있다., 개념적 ER 모델의 목적은 논리적 ER 모델의 집단 사이에서 master data entities를 위한 정형화된 메타데이터 commonality를 확립하는 것이다. 이러한 개념적 데이터 모델은 데이터 모델의 통합을 위한 기초로서 ER 모델 간의 commonality(속성의 공유) 관계를 형성하는데 사용될 수 있다.

Logical data model(논리적 데이터 모델)

A logical ER model does not require a conceptual ER model, especially if the scope of the logical ER model includes only the development of a distinct information system. The logical ER model contains more detail than the conceptual ER model. In addition to master data entities, operational and transactional data entities are now defined. The details of each data entity are developed and the entity relationships between these data entities are established. The logical ER model is however developed independent of technology into which it will be implemented.

논리적 모델은 개념적 ER 모델을 필요로 하지 않는다, 특히 만일에 논리적 모델의 범위가 단지 분명한 정보시스템의 개발만을 위한 것이라면. 논리적 ER 모델은 개념적 ER 모델보다 더 많은 내용이 포함된다. master data 객체에 따라서, 기능과 거래 데이터 객체가 이제 정의된다. 각 데이터 객체의 내역이 개발되며 이러한 데이터 객체간의 객체 관계가 확립된다. 논리적 ER 모델은 그렇지만 그것을 실현시킬 수 있는 기술과는 독립적이다.

Physical data model(물리적 데이터 모델)

One or more physical ER models may be developed from each logical ER model. The physical ER model is normally developed to be instantiated as a database. Therefore, each physical ER model must contain enough detail to produce a database and each physical ER model is technology dependent since each database management system is somewhat different.

하나 이상의 물리적 ER 모델이 각각의 논리적 ER 모델로부터 개발될 수 있다. 물리적 ER 모델은 보통 하나의 데이터베이스처럼 하나의 사례로 개발되고 있다. 그러므로 각각의 물리적 ER 모델은 데이터베이스를 생산할 수 있는 충분한 내역을 포함하고 있어야 하며 각각의 물리적 ER 모델은 각각의 데이터베이스 관리 시스템이 다소 차이가 나므로 기술적으로 독립성을 가져야 한다.

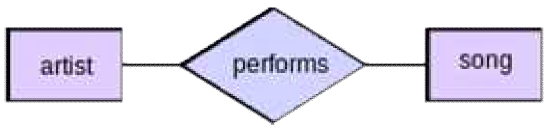
The physical model is normally forward engineered to instantiate the structural metadata into a database management system as relational database objects such as database tables, database indexes such as unique key indexes, and database constraints such as a foreign key constraint or a commonality constraint. The ER model is also normally used to design modifications to the relational database objects and to maintain the structural metadata of the database.

물리적 모델에서는 일반적으로 정형화된 메타데이트를 데이터베이스 테이블과 같은 관계형 데이터베이스 객체, unique key indexes와 같은 데이터베이스 색인, 외래키 제한 조건이나 commonality 제한조건과 같은 데이터베이스 제한조건에 맞게 사례별로 미리 만든다.

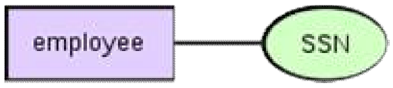
The first stage of information system design uses these models during the requirements analysis to describe information needs or the type of information that is to be stored in a database. The data modeling technique can be used to describe any ontology (i.e. an overview and classifications of used terms and their relationships) for a certain area of interest. In the case of the design of an information system that is based on a database, the conceptual data model is, at a later stage (usually called logical design), mapped to a logical data model, such as the relational model; this in turn is mapped to a physical model during physical design. Note that sometimes, both of these phases are referred to as "physical design". It is also used in database management system.

정보시스템 디자인의 첫 번째 단계에서는 데이터베이스에 저장되어야 하는 정보의 유형과 정보의 요구를 기술하기 위하여 필요사항을 분석하는 동안에 이러한 모델들을 사용한다. 데이터 모델링 기법은 관심의 특정 분야를 위하여 어떤 온톨로지(다시 말해서, 사용된 용어와 그것들의 관계에 대한 전반적 견해와 분류)를 기술하는데 이용될 수 있다. 데이터베이스를 근거로 하는 정보시스템을 디자인하는 경우에, 개념적 데이터 모델은 그것의 후반부에서(대체로 논리적 디자인이라고 부른다) 관계형 모델처럼 논리적 데이터 모델에 그려진다; 그 다음으로 이것은 물리적 디자인 동안에 물리적 모델로 그려진다. 주목할 것은 때때로 이러한 단계들 둘 다 ‘물리적 디자인’이라고 말한다. 또한 이것은 데이터베이스 관리 시스템에서도 사용된다.

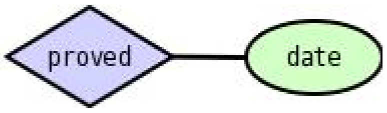
Entity-relationship modelling



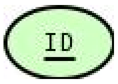
Two related entities



An entity with an attribute



A relationship with an attribute



Primary key

An entity may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of a domain. When we speak of an entity, we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world.
객체는 유일하게 식별가능하며 독립적인 존재로 인식되는 사물이라고 정의할 수 있다. 객체는 도메인의 복잡성에서부터 발생한 추상이다. 우리가 객체를 말할 때, 우리는 보통 실세계의 다른 요소와 구분할 수 있는 어떤 요소에 대하여 말하고 있다.

An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order. Although the term entity is the one most commonly used, following Chen we should really distinguish between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of an entity-type. Because the term entity-type is somewhat cumbersome, most people tend to use the term entity as a synonym for this term.

객체는 집, 차, 심지어 주택매매 또는 카 서비스의 물리적 개념, 또는 고객의 거래사항과 주문과 같은 개념일 수도 있다. 비로 객체라는 용어가 가장 널리 일반적으로 사용되고 있다 하더라도, Chen과 마찬가지로, 우리는 실재로 객체와 객체유형을 구분하여야 한다. 객체-유형은 범주이다. 엄격하게 말해서, 객체는 특정한 객체-유형의 한 사례이다. 대체로 객체-유형

에는 많은 사례가 존재한다. 객체-유형이라는 단어가 다소 모호하기 때문에, 대부분의 사람들은 이 용어의 이음동의어로 객체라는 용어를 사용하기도 한다.

Entities can be thought of as nouns. Examples: a computer, an employee, a song, a mathematical theorem. A relationship captures how entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Examples: an owns relationship between a company and a computer, a supervises relationship between an employee and a department, a performs relationship between an artist and a song, a proved relationship between a mathematician and a theorem.

객체는 명사로 여겨질 수 있다. 예를 들어, 컴퓨터, 종업원, 노래, 수학적 정리 등이다. 관계는 객체가 어떻게 서로 연결되어 있는가를 나타낸다. 관계는 두 개 이상의 명사를 링크시키는 동사로 여겨질 수 있다. 예를 들어, 회사와 컴퓨터 간의 소유 관계, 종업원과 부서 간의 감독 관계, 예술가와 노래 간의 연주 관계, 수학자와 정리간의 입증 관계 등이 있다.

The model's linguistic aspect described above is utilized in the declarative database query language ERROL, which mimics natural language constructs. ERROL's semantics and implementation are based on reshaped relational algebra (RRA), a relational algebra which is adapted to the entity-relationship model and captures its linguistic aspect.

위에서 설명한 모델의 언어학적 요소는 선언적 데이터베이스 쿼리 언어이며 자연적 언어 구조를 모방한 ERRO에서 사용되고 있다. ERROL의 어의와 실행은 재형성된 관계형 대수 (RRA)(객체-관계 모델에 적용되고 있으며 그것의 언어적 요소를 사용하고 있는 관계형 대수). 에 근거하고 있다.

Entities and relationships can both have attributes. Examples: an employee entity might have a Social Security Number (SSN) attribute; the proved relationship may have a date attribute. Every entity (unless it is a weak entity) must have a minimal set of uniquely identifying attributes, which is called the entity's primary key.

객체와 관계는 둘 다 속성을 가질 수 있다. 예를 들어 종업원 객체는 사회보장번호 속성을 가질 수 있다: 입증 관계는 날짜 속성을 가질 수 있다. 모든 객체는(약한 객체가 아니라면) 하나의 최소한 세트의 유일하게 식별 가능한 속성을 가져야만 한다. 이것을 그 객체의 으뜸키라고 부른다.

Entity-relationship diagrams don't show single entities or single instances of relations. Rather, they show entity sets and relationship sets. Example: a particular song is an entity. The collection of all songs in a database is an entity set. The eaten relationship between a child and her lunch is a single relationship. The set of all such child-lunch relationships in a database is a relationship set. In other words, a relationship set corresponds to a relation in mathematics, while a relationship corresponds to a member of the relation.

객체-관계 다이어그램은 단일 객체나 단일 관계의 사례를 보여주지 않는다. 그 보다는 이것들은 객체 세트와 관계 세트를 보여준다. 예를 들어, 특별한 song은 객체이다. 데이터베이스에서 모든 songs의 집단은 하나의 객체 세트이다. 어린이와 그녀의 점심 간의 eaten 관계는 단일 관계이다. 데이터베이스에서 그러한 모든 어린이-점심 관계의 세트는 하나의 관계 세트이다. 다른 말로 해서, 하나의 관계 세트는 수학에서 하나의 관계와 일치하지만, 하나의 관계는 그 관계의 한 멤버와 일치한다.

Certain cardinality constraints on relationship sets may be indicated as well.

관계 세트에서 어떤 cardinality 조건 또한 나타날 수 있다.

Mapping natural language

Chen proposed the following "rules of thumb" for mapping natural language descriptions into ER diagrams:

chen은 다음과 같은 ER 다이어그램에서 자연어를 기술하는 지도용 '경험의 법칙'을 제안하였다:

English grammar structure	**ER structure**
Common noun	Entity type
Proper noun	Entity
Transitive verb	Relationship type
Intransitive verb	Attribute type
Adjective	Attribute for entity
Adverb	Attribute for relationship

Physical view show how data is actually stored.

물리적 뷰는 어떻게 데이터가 실재적으로 저장되는지를 보여준다.

Relationships, roles and cardinalities

In Chen's original paper he gives an example of a relationship and its roles.

He describes a relationship "marriage" and its two roles "husband" and "wife". A person plays the role of husband in a marriage (relationship) and another person plays the role of wife in the (same) marriage. These words are nouns. That is no surprise; naming things requires a noun.

Chen의 원 본문에서, 그는 관계와 그것의 역할에 대한 예를 제시하였다. 그는 “결혼” 관계와 그것의 두 가지 역할로 “남편”과 “부인”을 설명하고 있다. 한 사람은 결혼(관계)에서 남편의 역할을 하고, 다른 사람은 (동일한) 결혼에서 부인 역할을 한다. 이들 단어들은 명사이다. 놀랄 것도 없이 사물을 명명하는데는 명사가 필요하다.

However as is quite usual with new ideas, many eagerly appropriated the new terminology but then applied it to their own old ideas. Thus the lines, arrows and crows-feet of their diagrams owed more to the earlier Bachman diagrams than to Chen's relationship diamonds. And they similarly misunderstood other important concepts. In particular, it became fashionable (now almost to the point of exclusivity) to "name" relationships and roles as verbs or phrases.

그렇지만 새로운 아이디어에 일상적이 듯이, 많은 노력이 새로운 용어에 적합하게 사용되었고 난 다음 그것의 오래된 아이디어에 응용되었다. 그러므로 이러한 다이어그램의 lines, arrows, crows-feet는 Chen의 관계형 다이어그램보다도 그 이전의 Bachman diagram에 더 많은 덕을 보고 있다. 특히, 동사와 어구로서 관계와 역할을 “명명”하는 것이 유행이 되었다(지금은 거의가 독점적으로 사용하고 있다).

Role naming

It has also become prevalent to name roles with phrases e.g. is-the-owner-of and is-owned-by etc. Correct nouns in this case are "owner" and "possession". Thus "person plays the role of owner" and "car plays the role of possession" rather than "person plays the role of is-the-owner-of" etc.

어구들로 역할을 명명하는 것이 널리 퍼졌다. 예를 들어, is-the-owner-of 와 is-owned-by 등이다. 이러한 경우에 정확한 명사는 “owner”와 “possession”이다. 그러므로 "person plays the role of is-the-owner-of" etc. 보다는 "person plays the role of owner" 와 "car plays the role of possession" 이다.

The use of nouns has direct benefit when generating physical implementations from semantic models. When a person has two relationships with car then it is possible to very simply generate names such as "owner_person" and "driver_person" which are immediately meaningful.

명사의 사용은 어의적 모델로부터 물리적 실행을 생산할 때 직접적인 도움이 되었다. 한 사람이 차와 함께 두 개의 관계를 가질 때, 그것은 즉시 그 의미를 알 수 있는

“owner_person”과 “driver_person”같이 매우 간단하게 이름을 만들 수 있다.

Cardinalities

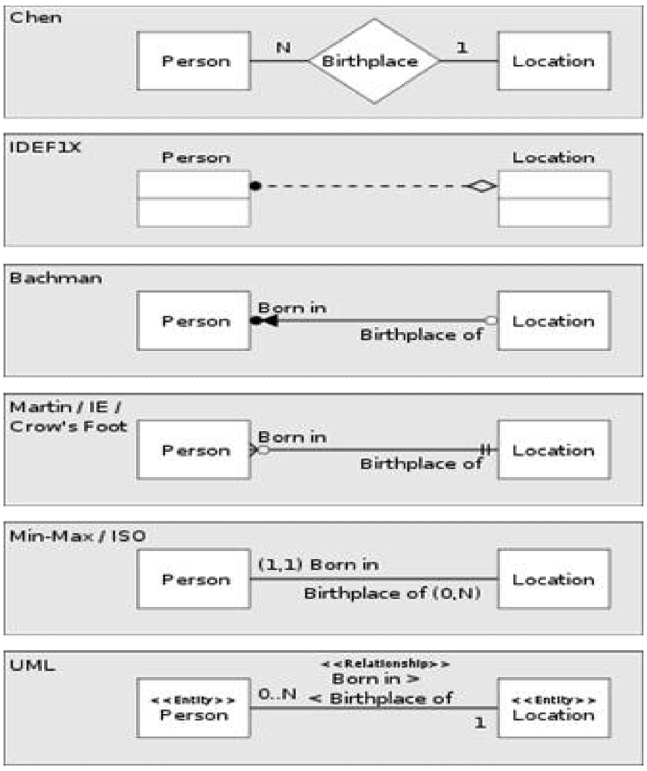
Modifications to the original specification can be beneficial. Chen described look-across cardinalities. As an aside, the Barker-Ellis notation, used in Oracle Designer, uses same-side for minimum cardinality (analogous to optionality) and role, but look-across for maximum cardinality (the crows foot).

원래 스펙의 변경은 유익할 수 있다. Chen은 look-across 카디낼러티를 설명하였다. 여담으로, Oracle Designer에서 사용한 Barker-Ellis 표기법은 최소 카디낼러티(선택적이라고 여겨지는)와 역할용으로 same-side를 사용하고 있지만 최대 카디낼러티용으로는 look - across를 사용하고 있다(the crows foot).

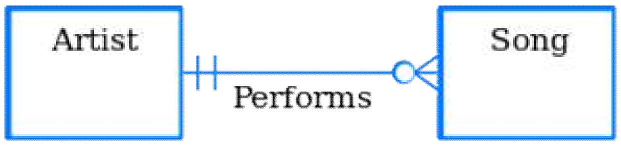
In Merise, Elmasri &Navathe and others there is a preference for same-side for roles and both minimum and maximum cardinalities. Recent researchers (Feinerer, Dullea et al.) have shown that this is more coherent when applied to n-ary relationships of order >2.

In Dullea et al. one reads "A 'look across' notation such as used in the UML does not effectively represent the semantics of participation constraints imposed on relationships where the degree is higher than binary.“

In Feinerer it says "Problems arise if we operate under the look-across semantics as used for UML associations. Hartmann investigates this situation and shows how and why different transformations fail." (Although the "reduction" mentioned is spurious as the two diagrams 3.4 and 3.5 are in fact the same) and also "As we will see on the next few pages, the look-across interpretation introduces several difficulties which prevent the extension of simple mechanisms from binary to n-ary associations.“



Various methods of representing the same one to many relationship. In each case, the diagram shows the relationship between a person and a place of birth: each person must have been born at one, and only one, location, but each location may have had zero or more people born at it.



Two related entities shown using Crow's Foot notation. In this example, an optional relationship is shown between Artist and Song; the symbols closest to the song entity represents "zero, one, or many", whereas a song has "one and only one" Artist. The former is therefore read as, an Artist (can) perform(s) "zero, one, or many" song(s).

Chen's notation for entity-relationship modeling uses rectangles to represent entity sets, and diamonds to represent relationships appropriate for first-class objects: they can have attributes and relationships of their own. If an entity set participates in a relationship set, they are connected with a line. Attributes are drawn as ovals and are connected with a line to exactly one entity or relationship set.

객체-관계 모델링의 Chen 표기법은 객체세트를 표현하기 위하여 사각형을, 그리고 자체적으로 속성과 관계를 가질 수 있는 제 1급 객체용으로 올바르게 관계를 표현하기 위하여 다이아몬드 형을 사용한다. 만일 객체 세트가 관계 세트에 참여한다면, 그것들은 선으로 연결된다. 속성은 타원형으로 그려지며 한 객체나 관계 세트에 직접적으로 선으로 연결된다.

Cardinality constraints are expressed as follows:
카디낼러지 제한조건은 다음과 같이 표현된다:

a double line indicates a participation constraint, totality or surjectivity(전사적인): all entities in the entity set must participate in at least one relationship in the relationship set;

복선은 참여조건을 나타낸다: totality 또는 surjectivity: 객체 세트에 있는 모든 객체는 관계 세트에 있는 적어도 하나의 관계에 참여하여야 한다.

an arrow from entity set to relationship set indicates a key constraint, i.e. injectivity: each entity of the entity set can participate in at most one relationship in the relationship set;

객체세트로부터 관계세트로의 화살표는 키 제한조건, 다시 말해서 injectivity를 나타낸다: 객체세트의 각 객체는 기껏해야 관계 세트에 있는 한 관계에 참여할 수 있다.

a thick line indicates both, i.e. bijectivity: each entity in the entity set is involved in exactly one relationship.
굵은 선은 양쪽, 다시 말해서 bijectivity를 나타낸다: 객체 세트에 있는 각 객체는 정확하게 하나의 관계에 포함되어야 한다.

an underlined name of an attribute indicates that it is a key: two different entities or relationships with this attribute always have different values for this attribute.

속성의 밑줄 이름은 그것이 키라는 것을 나타낸다: 두 개의 서로 다른 객체나 이러한 속성을 갖는 관계는 이 속성용으로 서로 다른 값을 항상 갖는다.

Attributes are often omitted as they can clutter up a diagram; other diagram techniques often list entity attributes within the rectangles drawn for entity sets.

속성은 종종 다이어그램을 어지럽히기 때문에 생략된다: 다른 다이어그램 기법은 종종 객체 세트용으로 그린 사각형 속에 객체 속성을 열거하기도 한다.

Related diagramming convention techniques:

Bachman notation
Barker's Notation
EXPRESS
IDEF1X
Martin notation
(min, max)-notation of Jean-Raymond Abrial in 1974
UML class diagrams
Merise
Object-Role Modeling

Crow's Foot Notation

Crow's Foot notation is used in Barker's Notation, SSADM and Information Engineering. Crow's Foot diagrams represent entities as boxes, and relationships as lines between the boxes. Different shapes at the ends of these lines represent the cardinality of the relationship.

까치발 표기법은 Barker's Notation, SSADM and Information Engineering에서 사용되고 있다. 까치발 다이어그램은 박스로 객체를 그리고 박스간의 선으로 관계를 표현한다. 이러한 선들의 끝에 있는 여러 가지 모양은 그 관계의 카디낼러티를 표현한다.

Crow's Foot notation was used in the consultancy practice CACI. Many of the consultants at CACI (including Richard Barker) subsequently moved to Oracle UK, where they developed the early versions of Oracle's CASE tools, introducing the notation to a wider audience. The following tools use Crow's Foot notation: ARIS, System Architect, Visio, PowerDesigner, Toad Data Modeler, DeZign for Databases, Devgems Data Modeler, OmniGraffle, MySQL Workbench and SQL Developer Data Modeler. CA's ICASE tool, CA Gen aka Information Engineering Facility also uses this notation. Historically XA Systems Silverrun-LDM (logical data model) also supported this notation.

ER diagramming tools

There are many ER diagramming tools. Free software ER diagramming tools that can interpret and generate ER models and SQL and do database analysis are MySQL Workbench (formerly DBDesigner), and Open ModelSphere (open-source). A freeware ER tool that can generate database and application layer code (webservices) is the RISE Editor. SQL Power Architect while proprietary also has a free community edition.

Proprietary ER diagramming tools are Avolution, ER/Studio, ERwin, DeZign for Databases, MagicDraw, MEGA International, ModelRight, Navicat Data Modeler, OmniGraffle, Oracle Designer, PowerDesigner, Prosa Structured Analysis Tool, Rational Rose, Software Ideas Modeler, Sparx Enterprise Architect, SQLyog, System Architect, Toad Data Modeler, and Visual Paradigm.

Free software diagram tools just draw the shapes without having any knowledge of what they mean, nor do they generate SQL. These include Creately, yEd, LucidChart, Calligra Flow, and Dia.

ER and semantic modelling

Peter Chen, the father of ER modelling said in his seminal paper:
"The entity-relationship model adopts the more natural view that the real world consists of entities and relationships. It incorporates some of the important semantic information about the real world.“

ER 모델링의 아버지인 Peter Chen은 자신의 세미나 논문에서 말했다: “객체-관계 모델은 실세계는 객체와 관계로 구성된다는 보다 자연적인 견해를 적용한 것이다. 이것은 실세계에 대한 어떤 중요한 어의적 정보를 품고 있다.”

He is here in accord with philosophic and theoretical traditions from the time of the Ancient Greek philosophers: Socrates, Plato and Aristotle (428 BC) through to modern epistemology(인식론), semiotics(기호학) and logic of Peirce, Frege and Russell. Plato himself associates knowledge with the apprehension of unchanging Forms (The forms, according to Socrates, are roughly speaking archetypes(원형) or abstract representations of the many types of things, and properties) and their relationships to one another. In his original 1976 article Chen explicitly contrasts entity-relationship diagrams with record modelling techniques:

"The data structure diagram is a representation of the organisation of records and is not an exact representation of entities and relationships.“

Several other authors also support his program:

A semantic model is a model of concepts, it is sometimes called a "platform independent model". It is an intensional model. At the latest since Carnap, it is well known that:

"...the full meaning of a concept is constituted by two aspects, its intension and its extension. The first part comprises the embedding of a concept in the world of concepts as a whole, i.e. the totality of all relations to other concepts. The second part establishes the referential meaning of the concept, i.e. its counterpart in the real or in a possible world".

An extensional model is one which maps to the elements of a particular methodology or technology, and is thus a "platform specific model". The UML specification explicitly states that associations in class models are extensional and this is in fact self-evident by considering the extensive array of additional "adornments" provided by the specification over and above those provided by any of the prior candidate "semantic modelling languages". "UML as a Data Modeling Notation, Part 2“

1) A Platform-Independent Model (PIM) in software engineering is a model of a software system or business system, that is independent of the specific technological platform used to implement it.

The term platform-independent model is most frequently used in the context of the model-driven architecture approach.Platform independent is program running on different processors like intel, AMD, Sun Micro Systems etc.; This model-driven architecture approach corresponds to the Object Management Group vision of Model Driven Engineering.

The main idea is that it should be possible to use a Model Transformation Language to transform a Platform-independent model into a Platform-specific model. In order to achieve this transformation, one can use a language compliant to the newly defined QVT standard. Examples of such languages are VIATRA or ATLAS Transformation Language. It means execution of the program is not restricted by the type of o/s used.

Limitations

ER models assume information content that can readily be represented in a relational database. They describe only a relational structure for this information:

ER 모델은 관계형 데이터베이스에서 쉽게 정보의 콘텐트를 표현할 수 있다고 가정한다.

They are inadequate for systems in which the information cannot readily be represented in relational form, such as with semi-structured data:

이것들은 유사정형화 데이터와 같이 관계형 형태로 쉽게 정보를 표현할 수 없는 시스템에는 부적당하다.

For many systems, possible changes to information contained are nontrivial and important enough to warrant explicit specification:

많은 시스템에서 내장된 정보의 잠재적 변화는 명확하게 스펙에서 보장하는 것 이상으로 주요하다.

Some authors have extended ER modeling with constructs to represent change, an approach supported by the original author; an example is Anchor Modeling. An alternative is to model change separately, using a process modeling technique. Additional techniques can be used for other aspects of systems. For instance, ER models roughly correspond to just 1 of the 14 different modeling techniques offered by UML:

어떤 저자들은 구조적으로 변화를 표현할 수 있도록 ER 모델링을 확장시켰으며 이러한 시도는 원 저자로부터 지지를 받았다. 그 예가 Anchor Modelling이다. 처리 모델링 기법을 사용하여 독립적으로 변화를 모델하는 것도 하나의 대안이다. 추가적이 기법들이 시스템의 또 다른 요소들로 사용될 수 있다. 예를 들어, ER 모델은 UML에서 제공하는 14가지의 다양한 모델링 기법의 단지 하나에 불과하다고 말할 수 있다.

ER modeling is aimed at specifying information from scratch. This suits the design of new, standalone information systems, but is of less help in integrating pre-existing information sources that already define their own data representations in detail.

ER 모델링은 scratch로부터 정보를 밝히는 것이 목표이다. 이것은 새롭고, 독립적인 정보시스템을 디자인하는 적격이지만, 상세하게 그 자체의 데이터 표현을 이미 정의하고 있는 기존의 정보자원을 통합하는 데는 도움이 되지 않는다.

Even where it is suitable in principle, ER modeling is rarely used as a separate activity. One reason for this is today's abundance of tools to support diagramming and other design support directly on relational database management systems. These tools can readily extract database diagrams that are very close to ER diagrams from existing databases, and they provide alternative views on the information contained in such diagrams.

비록 원칙적으로 적합하다로 하더라도, ER 모델링은 거의가 하나의 독립된 활동으로 사용되지는 않는다. 이러한 것의 한 가지 이유는 오늘 날 관계형 데이터베이스 관리 시스템에 직접적으로 프로그램과 디자인을 지원하는도구가 풍부해 졌기 때문이다. 이들 도구들에서 기존의 데이터베이스의 ER 다이어그램과 매우 비슷한 데이터베이스 다이어그램을 쉽게 발췌할 수 있으며, 그러한 다이어그램에 들어 있는 정보에 대하여 대안적 뷰를 제공한다.

In a survey, Brodie and Liu could not find a single instance of entity-relationship modeling inside a sample of ten Fortune 100 companies. Badia and Lemire blame this lack of use on the lack of guidance but also on the lack of benefits, such as lack of support for data integration.

서베이에서, Brodie와 Liu는 100개의 회사 샘플로부터 객체-관계 모델일의 사례를 찾을 수 없었다. Badia와 Lemire는 안내의 부족으로 인한 사용의 부족과 데이터 통합의 지원 부족으로 이익의 부족을 비난했다.

The enhanced entity-relationship model (EER modeling) introduces several concepts which are not present in ER modeling, which are closely related to object-oriented design, like is-a relationships.
is-a 관계와 같은 객체지향형과 밀접하게 관련된 개량형 EER이 ER 모델링에서 소개되지 않은 여러 가지 개념을 소개하였다.

For modelling temporal databases, numerous ER extensions have been considered. Similarly, the ER model was found unsuitable for multidimensional databases (used in OLAP applications); no dominant conceptual model has emerged in this field yet, although they generally revolve around the concept of OLAP cube (also known as data cube within the field).

임시적인 데이터베이스를 모델링하기 위하여, 수많은 ER 개량형이 고려되어 왔다. 유사하게도, ER 모델은 다차원적인 데이터베이스에서 부적당한 것으로 나타났다(OLAP 어플에서 사용하는데). 비록 이것들이 일반적으로 OLAP cube의 개념(이 분야에서는 data cube로 알려진)을 재탕하고 있다 하더라도 어떠한 지배적인 개념적 모델도 아직 이 분야에서는 나타나지 않고 있다.

1) In computer programming contexts, a data cube (or datacube) is a three- (or higher) dimensional array of values, commonly used to describe a time series of image data. A data cube is also used in the field of imaging spectroscopy, since a spectrally-resolved image is represented as a three-dimensional volume.

For a time sequence of color images, the array is generally four-dimensional, with the dimensions representing image X and Y coordinates, time, and RGB (or other color space) color plane.

Many high-level computer languages treat data cubes and other large arrays as single entities distinct from their contents. These languages, of which APL, IDL, NumPy, PDL, and S-Lang are examples, allow the programmer to manipulate complete film clips and other data en masse with simple expressions derived from linear algebra and vector mathematics. Some languages (such as PDL) distinguish between a list of images and a data cube, while many (such as IDL) do not.

Enhanced entity-relationship model

The EER model includes all of the concepts introduced by the ER model. Additionally it includes the concepts of a subclass and superclass (Is-a), along with the concepts of specialization and generalization. Furthermore, it introduces the concept of a union type or category, which is used to represent a collection of objects that is the union of objects of different entity types.

EER 모델은 ER 모델에서 소개된 모든 개념을 포함하고 있다. 추가적으로 이것에는 세분화와 일반화의 개념에 따라 하위 클라스와 상위 클라스(Is-a)의 개념이 포함되어 있다. 더구나, 이것에는 서로 다른 객체 유형의 객체 유니언으로 되어 있는 객체 집단을 표현하는데 사용되는 union type이나 category의 개념도 포함되어 있다.

Subclass and superclass

Entity type Y is a subtype (subclass) of an entity type X if and only if every Y is necessarily an X. A subclass entity inherits all attributes and relationships of its superclass entity. A subclass entity may have its own specific attributes and relationships (together with all the attributes and relationships it inherits from the superclass. One of the most common superclass examples is a vehicle with subclasses of Car and Truck. There are a number of common attributes between a car and a truck, which would be part of the Superclass, while the attributes specific to a car or a truck (such as max payload, truck type...) would make up two subclasses.

만일 또는 단지 모든 Y가 반드시 하나의 X라면 객체유형 Y는 객체유형 X의 하위유형(하위클라스)이다. 하위 클라스 객체는 그것의 상위 클라스 객체의 모든 속성과 관계를 포함하고 있다. 하위 클라스 객체는 그 자체적으로 특수한 속성과 관계를 가질 수 있다.(모든 속성과 관계는 모두 다 그것의 상위 클래스로부터 물려 받는다) 가장 일반적인 상위 클라스의 한 가지는 하위 클라스로 승용차와 화물차인 자동차이다. 승용차와 화물차 간에는 수많은 공동의 속성이 있으며, 상위 클라스의 일부가 될 수 있지만 승용차와 화물차에 한정된 속성(최대적재량, 화물차 유형 등)은 두 가지의 하위 클라스를 표현하게 된다.

Object model

An object database (also object-oriented database management system) is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object databases are different from relational databases which are table-oriented. Object-relational databases are a hybrid of both approaches.

객체 데이터베이스 도는 객체 지향적 데이터베이스 관리 시스템은 객체 지향형 프로그래미에서서 사용된 것처럼 객체의 형태로 정보를 표현한 데이터베이스 관리 시스템이다. 객체 데이터베이스는 테이블 지향적인 관계형 데이터베이스와 다르다. 객체-관계형 데이터베이스는 이 두 가지 방법의 혼합형이다.

Object-oriented database management systems (OODBMSs) combine database capabilities with object-oriented programming language capabilities. OODBMSs allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS. Because the database is integrated with the programming language, the programmer can maintain consistency within one environment, in that both the OODBMS and the programming language will use the same model of representation. Relational DBMS projects, by way of contrast, maintain a clearer division between the database model and the application.

객체지향형 데이터베이스 관리 시스템인 OODBMSs는 데이터베이스 기능과 객체지향형 프로그래밍 언어 기능이 결합된 것이다. OODBMSs는 객체지향형 프로그래머로 하여금 제품을 개발하고 객체처럼 그것을 저장하며 OODBMS에서 새로운 객체를 만들기 위하여 기존의 객체를 복사하거나 변경한다. 이 데이터베이스는 프로그래밍 언어와 통합되어 있기 때문에, 프로그래머는 OODBMS와 프로그래밍 언어 둘 다 표현하는데 있어서 똑같은 모델을 사용할 수 있는 한 개의 환경에서 일관성을 유지할 수 있다. 관계형 DBMSs 프로젝트는 대조적으로 데이터베이스 모델과 그 어플 간에 보다 명확한 구분이 유지된다.

As the usage of web-based technology increases with the implementation of Intranets and extranets, companies have a vested interest in OODBMSs to display their complex data. Using a DBMS that has been specifically designed to store data as objects gives an advantage to those companies that are geared towards multimedia presentation or organizations that utilize computer-aided design (CAD).

웹 의존형 기술이 intranets과 extranets의 실행과 더불어 증가함으로써, 기업들은 자신들의 복잡한 데이터를 보여주기 위하여 OODBMSs에 커다란 관심을 가지고 있다. 객체처럼 데이터를 저장하도록 특별하게 디자인 DBMS를 사용함으로서, CAD를 활용하여 멀티미디어의 표현과 조직 분야에 집중하는 이들 회사는 이득을 얻고 있다.

Document model

A document-oriented database is a computer program designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Document-oriented databases are one of the main categories of so-called NoSQL databases and the popularity of the term "document-oriented database" (or "document store") has grown with the use of the term NoSQL itself. In contrast to relational databases and their notions of "Relations" (or "Tables"), these systems are designed around an abstract notion of a "Document".

도큐멘트-지향적 데이터베이스는 유사-정형화 데이터로 알려진 도큐멘트-지향적 정보를 저장, 검색, 관리하기 위해 디자인된 컴퓨터 프로그램이다. 도큐멘트-지향적 데이터베이스는 소위 NoSQL 데이터베이스의 주요 범주 중의 하나이며, “도큐멘트-지향적 데이터베이스” 또는 “도큐멘트 스토어”라는 용어의 인기는 NoSQL 용어 그 자체의 사용과 함께 증가하고 있다. 관계형 데이터베이스와 그것의 “관계” 또는 “테이블”의 개념과는 대조적으로, 이 시스템들은 “도큐멘트”의 추상적 개념을 가지고 디자인한다.

Documents

The central concept of a document-oriented database is the notion of a Document. While each document-oriented database implementation differs on the details of this definition, in general, they all assume documents encapsulate and encode data (or information) in some standard formats or encodings. Encodings in use include XML, YAML, JSON, and BSON, as well as binary forms like PDF and Microsoft Office documents (MS Word, Excel, and so on).

도큐멘트 지향적 데이터베이스의 중심 개념은 도큐멘트의 개념이다. 각각의 도큐멘트-지향적 데이터베이스의 실행은 일반적으로 이러한 정의의 내용과는 차이가 나지만, 이것들은 모두 다 도큐멘트가 어떤 표준 포맷이나 암호화로 데이터나 정보를 감싸서 암호화한다고 생각한다. 사용가능한 암호화에는 XML, YAML, JSON, and BSON 뿐만 아니라 PDF와 Microsoft Office 도큐멘트와 같은 이진 형태(MS Word, Excell 등)가 포함된다.

Documents inside a document-oriented database are similar, in some ways, to records or rows in relational databases, but they are less rigid. They are not required to adhere to a standard schema, nor will they have all the same sections, slots, parts, or keys.

도큐멘트-지향적 데이터베이스 내부의 도큐멘트는 어떤 방법에서는 관계형 데이터베이스의 레코드나 로우와 비슷하지만, 엄격성이 다소 떨어진다. 이것들은 표준 스키마에 집착할 것을 요구 받지 않으며, 또한 모두가 동일 섹션, 슬로트, 파트, 또는 키를 갖지 않을 수도 있다.

For example, the following is a document:
다음의 예는 하나의 도큐멘트이다.

```
{
   FirstName:"Bob",
   Address:"5 Oak St.",
   Hobby:"sailing"
}
```

A second document might be:
두 번째 도큐멘트는 다음과 같을 수 있다:

```
{
   FirstName:"Jonathan",
   Address:"15 Wanamassa Point Road",
   Children:[
             {Name:"Michael", Age:10},
             {Name:"Jennifer", Age:8},
             {Name:"Samantha", Age:5},
             {Name:"Elena", Age:2}
           ]
}
```

These two documents share some structural elements with one another, but each also has unique elements. Unlike a relational database where every record contains the same fields, leaving unused fields empty; there are no empty 'fields' in either document (record) in the above example. This approach allows new information to be added to some records without requiring that every other record in the database share the same structure.

이들 두 개의 도큐멘트는 서로서로 몇 가지의 구조적 요소를 공유하고 있지만, 각각은 또한 유일한 요소들을 가지고 있다. 사용되지 않는 필드는 빈 공간으로 남겨 놓고 모든 레코드가 동일한 필드를 가지고 있는 관계형 데이터베이스와 달리, 이것에는 어떠한 빈 공간의 “필드”도 위의 예에서처럼 존재하지 않는다. 이러한 방법은 새로운 정보로 하여금 데이터베이스에 있는 모든 다른 레코드가 동일한 구조를 공유할 것을 요구하지 않고 어떤 레코드에 추가되는 것을 허락한다.

Keys

Documents are addressed in the database via a unique key that represents that document. This key is often a simple string, a URI, or a path. The key can be used to retrieve the document from the database. Typically, the database retains an index on the key to speed up document retrieval.

도큐멘트는 그 도큐멘트를 대표하는 유일한 키를 통해 데이터베이스에 자리를 잡는다. 이 키는 종종 간단한 문자열, URI, 또는 path 이다. 이 키는 데이터베이스로부터 그 도큐멘트를 검색하는데 사용될 수 있다. 전형적으로, 이 데이터베이스는 도큐멘트 검색의 속도를 높이기 위하여 그 키에 대한 색인을 가지고 있다.

Retrieval

Another defining characteristic of a document-oriented database is that, beyond the simple key-document (or key-value) lookup that can be used to retrieve a document, the database offers an API or query language that allows the user to retrieve documents based on their content. For example, you may want a query that retrieves all the documents with a certain field set to a certain value. The set of query APIs or query language features available, as well as the expected performance of the queries, varies significantly from one implementation to the next.

도큐멘트-지향형 데이터베이스의 또 다른 분명한 특징은 도큐멘트의 검색에 사용할 수 있는 간단한 key-document(or key-value) lookup 이외에도, 그 데이터베이스는 사용자로 하여금 콘텐트를 근거로 도큐멘트를 검색하도록 하는 API나 쿼리 언어를 제공한다는 것이다. 예를 들어, 여러분이 어떤 값으로 어떤 필드 세트를 가지고 있는 모든 도큐멘트를 검색할 수 있는 쿼리를 원할 수 있다. 이용 가능한 쿼리 APIs의 세트나 쿼리언어의 특징뿐만 아니라 그 쿼리에 대한 기대 성능은 매번 실행할 때마다 크게 차이가 난다.

Organization

Implementations offer a variety of ways of organizing documents, including notions of

다음과 같은 개념을 포함하여 도큐멘트를 조직하는 다양한 방법이 실행에서 제공된다.

Collections
Tags
Non-visible Metadata
Directory hierarchies
Buckets

Entity-attribute-value model

Entity-attribute-value model (EAV) is a data model to describe entities where the number of attributes (properties, parameters) that can be used to describe them is potentially vast, but the number that will actually apply to a given entity is relatively modest. In mathematics, this model is known as a sparse matrix. EAV is also known as object-attribute-value model, vertical database model and open schema.

EAV 모델은 객체를 기술하는데 사용될 수 있는 속성(성질, 변수)의 수가 잠재적으로 방대하지만 실제적으로 특정한 객체에 응용할 수 있는 그 수는 비교적 많지 않은 객체를 기술하는데 사용하는 데이터 모델이다. 수학에서, 이 모델은 sparse matrix로 알려져 있다. EAV는 또한 object-attribute-value model, vertical database model, open schema로도 알려져 있다.

1) Structure of an EAV table

This data representation is analogous to space-efficient methods of storing a sparse matrix, where only non-empty values are stored. In an EAV data model, each attribute-value pair is a fact describing an entity, and a row in an EAV table stores a single fact. EAV tables are often described as "long and skinny": "long" refers to the number of rows, "skinny" to the few columns.

이것의 데이터 표현은 단지 non-empty 값만을 저장하는 spare matrix를 저장하는 공간-효율성 방법이라 여겨지고 있다. EAV 데이터 모델에서, 각각의 속성-값 한 쌍은 객체를 기술하는 하나의 fact이며 EAV 테이블에 있는 로우는 단일 fact로 저장된다. EAV 테이블은 종종 “long and skinny”로 기술된다: “long”은 로우의 수를 말하며, “skinny”는 극소수의 컬럼을 말한다.

Data is recorded as three columns:

데이터는 3개의 컬럼에 저장된다:

The entity: the item being described.

객체: 기술대상 아이템

The attribute or parameter: a foreign key into a table of attribute definitions. At the very least, the attribute definitions table would contain the following columns: an attribute ID, attribute name, description, data type, and columns assisting input validation, e.g., maximum string length and regular expression, set of permissible values, etc.

속성 또는 매개변수: 속성 정의의 테이블로 들어가는 외래 키. 최소한으로, 속성 정의 테이블에는 다음과 같은 컬럼이 포함될 수 있다: 속성 ID, 속성 이름, 기술, 데이터 유형, 그리고 입력의 타당성을 지원하는 컬럼, 예를 들어, 최대 문자열 길이와 정규적인 표현, 허용 가능한 값의 세트 등.

The value of the attribute.
속성의 값.

Example

One example of EAV modeling in production databases is seen with the clinical findings (past history, present complaints, physical examination, lab tests, special investigations, diagnoses) that can apply to a patient. Across all specialities of medicine, these can range in the hundreds of thousands (with new tests being developed every month). The majority of individuals who visit a doctor, however, have relatively few findings.

No doctor would ever have the time to ask a patient about every possible finding. Instead, the doctor focuses on the primary complaints, and asks questions related to these. Still other questions are based on the responses to previously asked questions. Other questions or tests may be related to findings in the physical exam. The presence of certain findings automatically rules out many others — e.g., one would not consider pregnancy, and medical conditions associated with it if the patient were a male.

When the patient's record is summarized, one typically records "positive" findings — e.g., the presence of an enlarged and hardened liver — as well as "significant negatives" — e.g., the absence of signs suggestive of alcoholism (which is one of the causes to a hard, enlarged liver). In any case, one would not record the vast number of non-relevant findings that were not looked for or found in this particular patient.

Consider how one would try to represent a general-purpose clinical record in a relational database. Clearly creating a table (or a set of tables) with thousands of columns is not the way to go, because the vast majority of columns would be null. To complicate things, in a longitudinal medical record that follows the patient over time, there may be multiple values of the same parameter: the height and weight of a child, for example, change as the child grows. Finally, the universe of clinical findings keeps growing: for example, diseases such as SARS emerge, and new lab tests are devised; this would require constant addition of columns, and constant revision of the user interface. (The situation where the list of attributes changes frequently is termed "attribute volatility" in database parlance.)

The following shows a snapshot of an EAV table for clinical findings. The entries shown within angle brackets are references to entries in other tables, shown here as text rather than as encoded foreign key values for ease of understanding. They represent some details of a visit to a doctor for fever on the morning of 1/5/98. In this example, the values are all literal values, but these could also be foreign keys to pre-defined value lists; these are particularly useful when the possible values are known to be limited.

The entity. For clinical findings, the entity is the patient event: a foreign key into a table that contains at a minimum a patient ID and one or more time-stamps (e.g., the start and end of the examination date/time) that record when the event being described happened.

The attribute or parameter: a foreign key into a table of attribute definitions (in this example, definitions of clinical findings). At the very least, the attribute definitions table would contain the following columns: an attribute ID, attribute name, description, data type, units of measurement, and columns assisting input validation, e.g., maximum string length and regular expression, maximum and minimum permissible values, set of permissible values, etc.

The value of the attribute. This would depend on the data type, and we discuss how values are stored shortly.

The example below illustrates symptoms findings that might be seen in a patient with pneumonia.

(<patient XYZ, 1/5/98 9:30 AM>, <Temperature in degrees Fahrenheit>, "102")

(<patient XYZ, 1/5/98 9:30 AM>, <Presence of Cough>, "True")

(<patient XYZ, 1/5/98 9:30 AM>, <Type of Cough>, "With phlegm, yellowish, streaks of blood")

(<patient XYZ, 1/5/98 9:30 AM>, <Heart Rate in beats per minute>, "98")

...

EAV databases

The term "EAV database" refers to a database design where a significant proportion of the data is modeled as EAV. However, even in a database described as "EAV-based", some tables in the system are traditional relational tables.

As noted above, EAV modeling makes sense for categories of data, such as clinical findings, where attributes are numerous and sparse. Where these conditions do not hold, standard relational modeling (i.e., one column per attribute) is preferable; using EAV does not mean abandoning common sense or principles of good relational design. In clinical record systems, the subschemas dealing with patient demographics and billing are typically modeled conventionally. (While most vendor database schemas are proprietary, VistA, the system used throughout the United States Department of Veterans Affairs (VA) medical system, known as the Veterans Health Administration (VHA), is open-source and its schema is readily inspectable, though it uses a MUMPS database engine rather than a relational database.) As discussed shortly, an EAV database is essentially unmaintainable without numerous supporting tables that contain supporting metadata. The metadata tables, which typically outnumber the EAV tables by a factor of at least three or more, are typically standard relational tables. An example of a metadata table is the Attribute Definitions table mentioned above.

Star schema

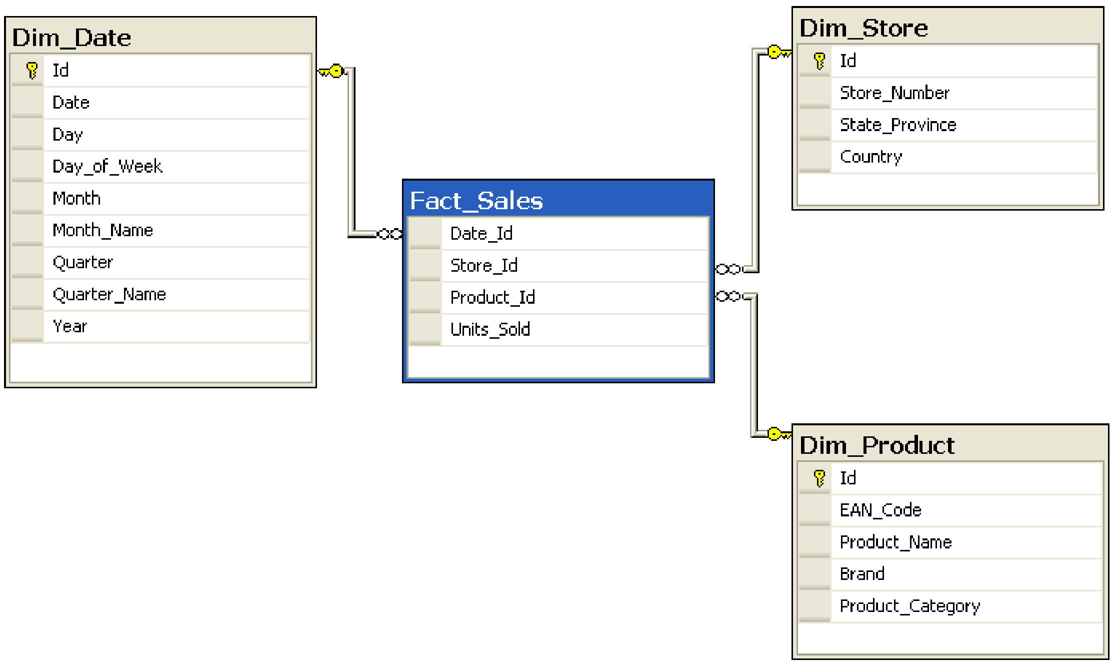
The Star Schema (also called star-join schema) is the simplest style of data mart schema. The star schema consists of one or more fact tables referencing any number of dimension tables. The star schema is an important special case of the snowflake schema, and is more effective for handling simpler queries.

스타 스키마(스타-조인 스키마라고도 부른다)는 데이터 mart 스키마의 가장 간단한 유형이다. 스타 스키마는 몇 개의 차원 테이블을 참조하는 하나 이상의 사실 테이블로 구성된다. 스타 스키마는 snowflake 스키마의 중요하고 특별한 경우이며, 보다 간단한 쿼리를 처리하는데 있어서 보다 효과적이다.

The star schema gets its name from the physical model's resemblance to a star with a fact table at its center and the dimension tables surrounding it representing the star's points.

스타 스키마는 그것의 중심에 사실 테이블이 있고 그 주위에 그 별의 포인트를 표현하고 있는 차원 테이블들로 이루어진 별과 같은 물리적인 모델의 형상에서부터 그 이름을 얻었다.

1) star schema Example



Star schema used by example query.

Consider a database of sales, perhaps from a store chain, classified by date, store and product. The image of the schema to the right is a star schema version of the sample schema provided in the snowflake schema article.

Fact_Sales is the fact table and there are three dimension tables Dim_Date, Dim_Store and Dim_Product.

Each dimension table has a primary key on its Id column, relating to one of the columns (viewed as rows in the example schema) of the Fact_Sales table's three-column (compound) primary key (Date_Id, Store_Id, Product_Id). The non-primary key Units_Sold column of the fact table in this example represents a measure or metric that can be used in calculations and analysis. The non-primary key columns of the dimension tables represent additional attributes of the dimensions (such as the Year of the Dim_Date dimension).

For example, the following query answers how many TV sets have been sold, for each brand and country, in 1997:

```
SELECT
        P.Brand,
        S.Country AS Countries,
        SUM(F.Units_Sold)
```

```
FROM Fact_Sales F
INNERJOIN Dim_Date D      ON F.Date_Id = D.Id
INNERJOIN Dim_Store S     ON F.Store_Id = S.Id
INNERJOIN Dim_Product P ON F.Product_Id = P.Id
WHERE
        D.YEAR=1997
AND P.Product_Category ='tv'
GROUPBY
        P.Brand,
        S.Country
```

An object-relational database combines the two related structures. Physical data models include:

객체-관계형 데이터베이스는 두 가지의 관련된 구조들로 결합된다. 물리적 데이터 모델에는 다음과 같은 것이 포함된다:

Inverted index

an inverted index (also referred to as postings file or inverted file) is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents. The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines. Several significant general-purpose mainframe-based database management systems have used inverted list architectures, including ADABAS, DATACOM/DB, and Model 204.

역색인(또는 포스팅 파일 또는 도치파일이라고도 함)은 단어와 숫자와 같은 콘텐트에서부터 데이터베이스 파일 또는 한 도큐멘트나 도큐멘트의 세트에 있는 그것의 위치까지 mapping을 저장하는 색인 데이터 구조이다. 역색인의 목적은 도큐멘트가 그 데이터베이스에 추가될 때 처리비용이 늘어나더라도 신속한 전문탐색이 가능하도록 하는 것이다. 도치파일은 색인이라기 보다는 데이터베이스 그 자체 파일일 수 있다. 이것은 탐색엔진에서 많은 예를 들고 있는 도큐멘트 검색 시스템에서 사용되는 가장 인기 있는 데이터 구조이다. 여러 가지의 중요한 범용의 대형 컴퓨터형 데이터베이스 관리 시스템이 ADABAS, DATACOM/DB, Model 204를 포함하여 도치 리스트 구조를 사용하고 있다.

There are two main variants of inverted indexes: A record level inverted index (or inverted file index or just inverted file) contains a list of references to documents for each word. A word level inverted index (or full inverted index or inverted list) additionally contains the positions of each word within a document. The latter form offers more functionality (like phrase searches), but needs more time and space to be created.

역색인에는 두 가지 중요한 유형이 있다: record level interted indes(또는 inverted file index 또는 단지 inverted file)은 각 단어에 맞는 도큐멘트들에 대한 참고 리스트를 가지고 있다. word level inverted index(또는 full inverted index 또는 inverted list)는 추가로 도큐멘트에 각 단어의 위치를 포함하고 있다. 후자는 더 많은 기능성(어구 탐색과 같은)을 제공하지만 만드는데 시간과 공간이 더 많이 요구된다.

Example

Given the texts
다음과 같은 텍스가 주어졌을 때,

T[0] = "it is what it is"
T[1] = "what is it"
T[2] = "it is a banana"

we have the following inverted file index (where the integers in the set notation brackets refer to the indexes (or keys) of the text symbols, T[0], T[1] etc.):

우리는 이 텍스트의 심벌에 대한 다음과 같은 역색인 파일을 갖는다.(집합표기 모난 괄호 속에 있는 정수들은 텍스트 심벌, T[0], T[1] 등)의 색인(또는 키)을 말한다.)

"a":	{2}
"banana":	{2}
"is":	{0, 1, 2}
"it":	{0, 1, 2}
"what":	{0, 1}

A term search for the terms "what", "is" and "it" would give the set
"what", "is", "it"의 용어 탐색을 통해 다음과 같은 집합을 얻을 수 있다.

$$\{0,1\} \cap \{0,1,2\} \cap \{0,1,2\} = \{0,1\}.$$

With the same texts, we get the following full inverted index, where the pairs are document numbers and local word numbers. Like the document numbers, local word numbers also begin with zero. So, "banana": {(2, 3)} means the word "banana" is in the third document (T[2]), and it is the fourth word in that document (position 3).

동일한 텍스트를 가지고 우리는 각각의 쌍이 도큐멘트 번호와 해당 단어 번호인 다음과 같은 full inverted index을 얻는다. 도큐멘트 번호와 마찬가지로 해당 단어 번호 역시 0으로 시작한다. 그러므로 “banana”:{[2,3]}은 단어“banana”는 3번째 도큐멘트(T[2])에 있으며 이것은 그 도큐멘트에서 4번째 단어이다(position 3)>

"a":	{(2, 2)}
"banana":	{(2, 3)}
"is":	{(0, 1), (0, 4), (1, 1), (2, 1)}
"it":	{(0, 0), (0, 3), (1, 2), (2, 0)}
"what":	{(0, 2), (1, 0)}

If we run a phrase search for "what is it" we get hits for all the words in both document 0 and 1. But the terms occur consecutively only in document 1.

우리가 어구탐색 “what is it”을 실행한다면, 우리는 도큐멘트 0과 1 양쪽에서 모든 단어에 대한 결과를 얻는다.

Applications

The inverted index data structure is a central component of a typical search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs. Once a forward index is developed, which stores lists of words per document, it is next inverted to develop an inverted index. Querying the forward index would require sequential iteration through each document and to each word to verify a matching document. The time, memory, and processing resources to perform such a query are not always technically realistic. Instead of listing the words per document in the forward index, the inverted index data structure is developed which lists the documents per word.

도치색인 데이터 구조는 전형적인 탐색엔진 색인 알고리즘의 핵심 구성요소이다. 탐색엔진의 실행 목적은 쿼리의 속도를 최적화시키는 것이다: 단어 X의 빈도를 갖고 있는 도큐멘트를 찾는 것이다. 일잔 도큐멘트당 단어의 리스트를 저장하고 있는 forward index이 개발되었다면, 그다음은 도치색인을 개발하기 위하여 도치하는 것이다. forward 색인에 쿼리하는 것은 매칭되는 도큐멘트를 입증하기 위하여 각각의 도큐멘트에 있는 각각의 단어를 순차적 반복할 것을 요구한 다. 그러한 쿼리를 수행하기 위하여 시간, 메모리, 그리고 처리 자원들이 항상 기술적으로 현실화되지는 않는다. forward 색인에 있는 도큐멘트당 단어를 리스트하는 대신에,

도치 색인 데이터 구조는 단어당 도큐멘트를 리스트 하도록 개발되었다.

With the inverted index created, the query can now be resolved by jumping to the word id (via random access) in the inverted index. In pre-computer times, concordances to important books were manually assembled. These were effectively inverted indexes with a small amount of accompanying commentary that required a tremendous amount of effort to produce.

도치색인이 개발됨으로써, 쿼리는 이제 도치색에 있는 단어 id(랜덤 접근을 통하여)로 점프함으로써 해결 가능하게 되었다. 컴퓨터 시대 이전에, 중요한 책에 대한 용어색인은 수작업으로 수집되었다. 이것들은 생산하는데 커다란 노력이 요구되는 코멘트를 동반하는 소량의 효과적인 도치색인들이었다.

Flat file

A flat file database describes any of various means to encode a database model (most commonly a table) as a single file. A flat file can be a plain text file or a binary file. There are usually no structural relationships between the records.

플랫 파일 데이터베이스는 하나의 파일처럼 데이터베이스 모델(가장 일반적으로 말해서 하나의 테이블)을 암호화하는 여러 가지 수단들 중의 하나이다. 플랫 파일은 plain text file 이거나 binary file일 수 있다. 대체로 레코드들 간에는 어떠한 구조적 관계도 갖고 있지 않다.

Overview

Plain text files usually contain one record per line, There are different conventions for depicting data. In comma-separated values(CSV) and delimiter-separated values files, fields can be separated by delimiters such as comma or tab characters. In other cases, each field may have a fixed length; short values may be padded with space characters. Extra formatting may be needed to avoid delimiter collision. More complex solutions are markup languages and programming languages.

plain text files은 대체로 줄 당 한 개의 레코드를 가지고 있다. 데이터를 표현하는 데는 여러 가지 규정이 있다. comma-separated values 와 delimiter-separated values 파일들에 있어서 각 필드는 고정장일 수도 있다; 짧은 값들은 space 문자와 같이 첨부될 수 있다. 특별한 포맷팅이 delimiter collision을 피하기 위하여 요구되기도 한다. 보다 복잡한 해결책들은 markup language와 programming languages이다.

1) A comma-separated values (CSV) (also sometimes called character-separated values, because the separator character does not have to be a comma) file stores tabular data (numbers and text) in plain-text form. Plain text means that the file is

a sequence of characters, with no data that has to be interpreted instead, as binary numbers. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by some other character or string, most commonly a literal comma or tab. Usually, all records have an identical sequence of fields.

A general standard for the CSV file format does not exist, but RFC 4180 provides a de facto standard for some aspects of it.

2) Formats that use delimiter-separated values (also DSV) store two-dimensional arrays of data by separating the values in each row with specific delimiter characters. Most database and spreadsheet programs are able to read or save data in a delimited format.

Delimited formats

Any character may be used to separate the values, but the most common delimiters are the comma, tab, and colon. The vertical bar (also referred to as pipe) and space are also sometimes used. In a comma-separated values (CSV) file the data items are separated using commas as a delimiter, while in a tab-separated values (TSV) file, the data items are separated using tabs as a delimiter. Column headers are sometimes included as the first line, and each subsequent line is a row of data. The lines are separated by newlines.

For example, the following fields in each record are delimited by commas, and each record by newlines:

```
"Date","Pupil","Grade"
"25 May","Bloggs, Fred","C"
```

"25 May","Doe, Jane","B"
"15 July","Bloggs, Fred","A"
"15 April","Muniz, Alvin ""Hank""","A"

Note the use of the double quote to enclose each field. This prevents the comma in the actual field value (Bloggs, Fred; Doe, Jane and etc.) from being interpreted as a field separator. This necessitates a way to "escape" the field wrapper itself, in this case the double quote; it is customary to double the double quotes actually contained in a field as with those surrounding "Hank". In this way, any ASCII text including newlines can be contained in a field.

ASCII includes several control characters that are intended to be used as delimiters. They are: 28 file separator, 29 group separator, 30 record separator, 31 unit separator. Use of these characters has not achieved widespread adoption; some systems have replaced their control properties with more accepted controls such as CR/LF and TAB.

3) Delimiter collision

When using quoting, if one wishes to represent the delimiter itself in a string literal, one runs into the problem of delimiter collision. For example, if the delimiter is a double quote, one cannot simply represent a double quote itself by the literal """ as the second quote in interpreted as the end of the string literal, not as the value of the string, and similarly one cannot write "This is "in quotes", but invalid." as the middle quoted portion is instead interpreted as outside of quotes. There are various solutions, the most general-purpose of which is using escape sequences, such as "\"" or "This is \"in quotes\" and properly escaped.", but there are many other solutions.

Using delimiters incurs some overhead in locating them every time they are processed (unlike fixed-width formatting), which may have performance implications. However, use of character delimiters (especially commas) is also a crude form of data compression which may assist overall performance by reducing data volumes — especially for data transmission purposes. Use of character delimiters which include a length component (Declarative notation) is comparatively rare but vastly reduces the overhead associated with locating the extent of each field.

delimiters를 사용하는 것은 성능과 관련해서 되기도 하는 그것들을 처리(고정장 길이의 포매팅과는 달리)할 때마다, 그것이 자리를 잡는데 어느 정도의 추가비용이 발생한다. 그렇지만 또한 character delimiters(특히 쉼표)를 사용하는 것은 데이터의 크기를 줄임으로써 - 특히 데이터 전송 목적으로 - 전반적인 성능에 도움을 주는 데이터 압축의 원형이다. 길이 요소

(Declarative notation)에 포함되어 있는 character delimiters의 사용은 비교적 드물지만 각 필드의 외양으로 인하여 발생하는 추가비용을 크게 줄인다.

Typical examples of flat files are /etc/passwd and /etc/group on Unix-like operating systems. Another example of a flat file is a name-and-address list with the fields Name, Address, and Phone Number.

전형적인 플랫 파일의 예는 유닉스와 같은 운영체제에서의 /etc/passwd 와 /etc/group 이다. 또 다른 예는 필드 Name, Address, 그리고 Phone Number를 갖고 있는 name-and-address list 이다.

A list of names, addresses, and phone numbers written by hand on a sheet of paper is a flat file database. This can also be done with any typewriter or word processor. A spreadsheet or text editor program may be used to implement a flat file database, which may then be printed or used online for improved search capabilities.

시트나 페이퍼에 직접 기록한 names, addresses, phone numbers의 리스트는 flat file 데이터베이스이다. 이것은 또한 타자기나 워드프로세서에 의해 작성될 수 있다. 스프레드시트나 텍스트 편집 프로그램은 플랫파일 데이터베이스를 기동시키기 위하여 사용될 수 있다. 그리고 나서 출력시킬 수도 있고 또는 개선된 탐색성능을 가지고 온라인으로 사용할 수도 있다.

Example database

The following example illustrates the basic elements of a flat-file database. The data arrangement consists of a series of columns and rows organized into a tabular format. This specific example uses only one table. The columns include: name (a person's name, second column); team (the name of an athletic team supported by the person, third column); and a numeric unique ID, (used to uniquely identify records, first column).

다음의 예는 플랫 파일 데이터베이스의 기본적 요소를 나타내고 있다. 데이터 배열은 테이블 포맷으로 조직된 일련의 컬럼과 로우로 이루어져 있다. 이 특별한 예에서는 단지 하나의 테이블 만을 사용한다. 컬럼들에는 name(사람의 이름, 2번째 컬럼); team(그 사람에 의해 지원을 받는 운동팀의 이름, 3번째 컬럼); 그리고 숫자로 된 unique ID(유일하게 레코드를 식별하는데 사용된다, 1번째 컬럼)이 포함되어 있다.

Here is an example textual representation of the described data:

id	name	team
1	Amy	Blues

id	name	team
2	Bob	Reds
3	Chuck	Blues
4	Dick	Blues
5	Ethel	Reds
6	Fred	Blues
7	Gilly	Blues
8	Hank	Reds

This type of data representation is quite standard for a flat-file database, although there are some additional considerations that are not readily apparent from the text:

비록 텍스트로 표현하는 것이 분명히 쉽지 않다는 몇 가지 추가적인 고려사항이 있더라도, 이러한 유형의 데이터 표현은 플랫-파일 데이터베이스용으로는 알맞은 표준이다.

Data types: each column in a database table such as the one above is ordinarily restricted to a specific data type. Such restrictions are usually established by convention, but not formally indicated unless the data is transferred to a relational database system.

데이터 유형: 데이터베이스 테이블에 있는 각 컬럼은 보통은 특별한 데이터 유형으로 제한되어 있다. 그 같은 제한은 대체로 규정에서 마련되지만, 만일 그 데이터가 관계형 데이터베이스 시스템으로 전이되지 않는다면 공식적으로 지정되지 않는다.

Separated columns: In the above example, individual columns are separated using whitespace characters. This is also called indentation or "fixed-width" data formatting. Another common convention is to separate columns using one or more delimiter characters. More complex solutions are markup and programming languages.

분리 컬럼: 위의 예에서처럼, 각 컬럼들은 whitespace 문자를 사용하여 분리된다. 이것은 또한 indentation 또는 “fixed-wideth” data formatting이라고 부른다. 또 다른 일반적인 규정은 하나 이상의 delimiter 문자를 사용하여 컬럼을 분리하는 것이다. 보다 복잡한 해결책은 markup과 programming 언어들이다.

Relational algebra: Each row or record in the above table meets the standard definition of a tuple under relational algebra (the above example depicts a series of 3-tuples). Additionally, the first row specifies the field names that are associated with the values of each row.

관계 대수: 위의 테이블에 있는 각각의 로우나 레코드는 관계형 대수의 tuple의 표준 정

의를 충족시키고 있다(위의 예에서는 일련의 3-tuples를 표현하고 있다). 추가적으로, 첫 번째 로우는 각 로우의 값들과 결합되어있는 name 필드를 나타내고 있다.

The main application of relational algebra is providing a theoretical foundation for relational databases, particularly query languages for such databases, chief among which is SQL.

관계형 대수의 주요 어플은 관계형 데이터베이스, 특히 그러한 데이터베이스용의 쿼리 언어 중에서 대표격인 SQL의 이론적 근거를 제공하는 것이다.

1) relational algebra is an offshoot of first-order logic and of algebra of sets concerned with operations over finitary relations, usually made more convenient to work with by identifying the components of a tuple by a name (called attribute) rather than by a numeric column index, which is called a relation in database terminology.

2) Relational calculus consists of two calculi, the tuple relational calculus and the domain relational calculus, that are part of the relational model for databases and provide a declarative way to specify database queries. This in contrast to the relational algebra which is also part of the relational model but provides a more procedural way for specifying queries.

The relational algebra might suggest these steps to retrieve the phone numbers and names of book stores that supply Some Sample Book:

1. Join book stores and titles over the BookstoreID.
2. Restrict the result of that join to tuples for the book Some Sample Book.
3. Project the result of that restriction over StoreName and StorePhone.

The relational calculus would formulate a descriptive, declarative way:

Get StoreName and StorePhone for supplies such that there exists a title BK with the same BookstoreID value and with a BookTitle value of Some Sample Book.

The relational algebra and the relational calculus are essentially logically equivalent: for any algebraic expression, there is an equivalent expression in the calculus, and vice versa. This result is known as Codd's theorem.

Database management system: Since the formal operations possible with a text file are usually more limited than desired, the text in the above example would ordinarily represent an intermediary state of the data prior to being transferred into a database management system.

데이터베이스 관리 시스템: 텍스트 파일과 함께 공식적인 운영의 가능성이 대체로 원하는 것보다 매우 제한적이므로, 위의 예에 있는 텍스트는 보통 데이터베이스 관리 시스템에 전달되기 전에 데이터의 중간매체 상태로 표현될 수 있다.

Other models include:

Associative model

The associative model of data is an alternative data model for database systems. Other data models, such as the relational model and the object data model, are record-based. These models involve encompassing attributes about a thing, such as a car, in a record structure. Such attributes might be registration, colour, make, model, etc. In the associative model, everything which has "discrete independent existence" is modeled as an entity, and relationships between them are modeled as associations. The granularity at which data is represented is similar to schemes presented by Chen (Entity-relationship model); Bracchi, Paolini and Pelagatti (Binary Relations); and Senko (The Entity Set Model).

데이터의 associative model은 데이터베이스 시스템용의 대안적 데이터 모델이다. 관계형 모델과 객체 데이터 모델과 같은 다른 데이터 모델들은 레코드 의존적이다. 이러한 모델들은 레코드 구조 속에서 자동차와 같은 하나의 사물에 대한 속성들을 동반한다. 그 같은 속성들은 등록, 색깔, 제조, 모델 등일 수 있다. associative 모델에서, "분명한 독립적 존재"를 갖고 있는 모든 것은 하나의 객체로 모델화 되며, 그것들 간의 관계는 association처럼 모델화 된다. 그러한 데이터를 표현하는 granularity는 Chen(객체-관계 모델); Bracchi, Paolini와 Pelagatti(바이너리 관계); Senko(객체 세트 모델)에서 제시된 스키마와 비슷하다.

Multidimensional model

Multidimensional structure is defined as "a variation of the relational model that uses multidimensional structures to organize data and express the relationships between data". The structure is broken into cubes and the cubes are able to store and access data within the confines of each cube. "Each cell within a multidimensional structure contains aggregated data related to elements along each of its dimensions". Even when data is manipulated it remains easy to access and continues to constitute a compact database format. The data still remains interrelated. Multidimensional structure is quite popular for analytical databases that use online analytical processing (OLAP) applications.

다차원 구조는 "데이터를 조직하고 데이터 간의 관계를 표현하는데 있어서 다차원 구조를 사용하는 관계형 모델의 변종"으로 정의되고 있다. 그 구조는 cubes로 쪼개지며 cubes는 각

큐브의 범위 내에서 데이터를 저장하고 접근할 수 있다. "다차원 구조에 있는 각 셀은 각각의 차원에 따라 존재하는 요소들과 관련돼서 수집된 데이터를 포함하고 있다". 심지어 데이터가 다루어질 때, 이것은 접근하기 쉬우며 압축 데이터베이스 포맷을 지속적으로 구성한다. 그 데이터는 아직까지 상호연관성을 갖고 있다. 다차원 구조는 온라인 분석 처리(OLAP) 어플을 사용하는 분석적 데이터베이스에서 매우 인기가 높다.

1) The core of any OLAP system is an OLAP cube (also called a 'multi-dimensional cube' or a hypercube). It consists of numeric facts called measures which are categorized by dimensions. The measures are placed at the intersections of the hypercube, which is spanned by the dimensions as a Vector space. The usual interface to manipulate an OLAP cube is a matrix interface like Pivot tables in a spreadsheet program, which performs projection operations along the dimensions, such as aggregation or averaging.

The cube metadata is typically created from a star schema or snowflake schema or fact constellation of tables in a relational database. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables.

Each measure can be thought of as having a set of labels, or meta-data associated with it. A dimension is what describes these labels; it provides information about the measure.

A simple example would be a cube that contains a store's sales as a measure, and Date/Time as a dimension. Each Sale has a Date/Time label that describes more about that sale.

Any number of dimensions can be added to the structure such as Store, Cashier, or Customertime_id by adding a foreign key column to the fact table. This allows an analyst to view the measures along any combination of the dimensions.

For example:

Sales Fact Table

sale_amount	time_id
2008.10	1234

Time Dimension

time_id	timestamp
1234	20080902 12:35:43

Analytical databases use these databases because of their ability to deliver answers to complex business queries swiftly. Data can be viewed from different angles, which gives a broader perspective of a problem unlike other models.

분석적 데이터베이스는 이러한 데이터베이스를 사용하는데 그 이유는 복잡한 경영상의 쿼리에 대해 신속하게 해답을 제공할 수 있는 능력 때문이다. 데이터는 다른 모델과 달리 어떤 문제를 보는데 보다 넓은 시각으로 볼 수 있도록 다양한 각도를 제공할 수 있다.

Multivalue model

MultiValue is a type of NoSQL and multidimensional database, typically considered synonymous with PICK, a database originally developed as the Pick operating system. MultiValue databases include commercial products from Rocket Software, TigerLogic, jBASE, Revelation, Ladybridge, InterSystems, Northgate Information Solutions and other companies. These databases differ from a relational database in that they have features that support and encourage the use of attributes which can take a list of values, rather than all attributes being single-valued. They are often categorized with MUMPS within the category of post-relational databases, although the data model actually pre-dates the relational model. Unlike SQL-DBMS tools, most MultiValue databases can be accessed both with or without SQL.

MultiValue는 NoSQL과 다차원 데이터베이스의 한 유형이며, 전형적으로 Pick 운영체계용으로 원래 개발된 PICK와 동의어로 여겨지고 있다. MultiValue 데이터베이스는 Rocket Software, TigerLogic, jBASE, Revelation, Ladybridge, InterSystems, Northgate Information Solutions and other companies에서 만든 상업적 제품이 포함되어 있다. 이러한 데이터베이스들은 단일 값으로된 모든 속성보다도 값의 리스트를 가질 수 있는 속성들의 사용을 지원하는 특징을 갖고 있는 관계형 데이터베이스와 다르다. 이것들은 비록 그 데이터 모델이 실재로 관계형 모델보다는 앞선 시기에 만들어졌지만, 가끔 post-relational 데이터베이스의 범주에 포함되는 MUMPS로 분류되기도 한다. SQL-DBMS 도구와 달리, 대부분의 MultiValue 데이터베이스는 SQL로 또는 이것 없이도 접근할 수 있다.

1) MUMPS (Massachusetts General Hospital Utility Multi-Programming System, later: 'Multi-User Multi-Programming System') or alternatively M, is a general-purpose computer programming language that provides ACID (Atomic, Consistent, Isolated, and Durable) transaction processing. Its most unique and differentiating feature is its "built-in" database, enabling high-level access to disk storage using simple symbolic program variables (subscripted arrays), similar to the variables used by most languages to access main memory.

The M database is a key-value database engine optimized for high-throughput transaction processing. As such it is in the class of "schema-less", "schema-free," or NoSQL databases. Internally, M stores data in multidimensional hierarchical sparse arrays (also known as key-value nodes, sub-trees, or associative memory). Each array may have up to 32 subscripts, or dimensions. A scalar can be thought of as an array element with zero subscripts. Nodes with varying numbers of subscripts (including one node with no subscripts) can freely co-exist in the same array.

Perhaps the most unusual aspect of the M language is the notion that the database is accessed through variables, rather than queries or retrievals. This means that accessing volatile memory and non-volatile storage use the same basic syntax, enabling a function to work on either local (volatile) or global (non-volatile) variables. Practically, this provides for extremely high performance data access.

Semantic model

A semantic data model in software engineering has various meanings:

씨멘틱 데이터 모델은 소프트웨어 공학에서 여러 가지의 의미를 가지고 있다:

1. It is a conceptual data model in which semantic information is included. This means that the model describes the meaning of its instances. Such a semantic data model is an abstraction that defines how the stored symbols (the instance data) relate to the real world.
이것은 씨멘틱 정보가 포함되어 있는 개념적 데이터 모델이며, 이 모델은 그것의 경우에 대한 의미를 기술한다는 의미를 가지고 있다. 이러한 씨멘틱 데이터 모델은 저장된 심벌(경우의 데이터)가 실세계와 어떻게 관련되어 있는지를 설명하는 추상적 개념이다.

2. It is a conceptual data model that includes the capability to express information that enables parties to the information exchange to interpret meaning (semantics) from the instances, without the need to know the meta-model. Such semantic models are fact oriented (as opposed to object oriented). Facts are typically expressed by binary relations between data elements, whereas higher order relations are expressed as collections of binary relations. Typically binary relations have the form of triples: Object-Relation Type-Object. For example: the Eiffel Tower <is located in> Paris.

메타-모델을 알려고 하는 필요성 없이 그것의 경우로부터 의미(어의)를 해석하기 위하여 parties로 하여금 정보교환이 가능하도록 정보를 표현하는 능력을 가지고 있는 개념적 데이터 모델이다. 이러한 씨멘틱 모델은 사실 지향적이다(객체 지향적과는 반대인). 사실은 전형적으로 데이터 요소 간의 이진관계에 의해 표현된다. 반면에 고차원의 order 관계는 이진 관계의 집단으로 표현된다. 전형적으로 이진관계는 triples의 형태: Object-Relation Type-Object를

갖는다. 예: the Eiffel Tower <is located in> Paris.

Typically the instance data of semantic data models explicitly include the kinds of relationships between the various data elements, such as <is located in>. To interpret the meaning of the facts from the instances it is required that the meaning of the kinds of relations (relation types) is known. Therefore, semantic data models typically standardise such relation types. This means that the second kind of semantic data models enable that the instances express facts that include their own meaning. The second kind of semantic data models are usually meant to create semantic databases. The ability to include meaning in semantic databases facilitates building distributed databases that enable applications to interpret the meaning from the content. This implies that semantic databases can be integrated when they use the same (standard) relation types. This also implies that in general they have a wider applicability than relational or object oriented databases.

전형적으로 씨멘틱 데이터 모델의 경우 데이터는 분명하게 다양한 데이터 요소 간의 복수의 종류의 관계(<is located in>처럼)를 포함하고 있다. 그러한 경우들로부터 사실의 의미를 해석하기 위하여, 필요한 것은 관계의 종류(관계 유형)에 대한 의미를 파악하는 것이다. 그러므로 씨멘틱 데이터 모델은 전형적으로 그러한 관계 유형을 표준으로 삼는다. 이것이 의미하는 것은 2번째 종류의 씨멘틱 데이터 모델이 경우가 자기 자신의 의미를 포함하여 사실을 표현할 수 있다는 것이다. 2번째 종류의 씨멘틱 데이터 모델은 대체로 씨멘틱 데이터베이스를 만드는 것을 말한다. 씨멘틱 데이터베이스에서 의미를 포함할 수 있는 능력은 어플로 하여금 콘텐트로부터 의미를 해석하도록 하는 분산형 데이터베이스를 구축하는 것을 용이하게 한다. 이것이 의미하는 것은 씨멘틱 데이터베이스는 그것들이 동일한 또는 표준 관계 유형을 사용할 때 통합될 수 있다는 것이다. 또한 이것이 의미하는 것은 일반적으로 그것들은 관계형 또는 객체 지향형 데이터베이스보다 더욱 다양한 응용성을 가지고 있다는 것이다.

Overview

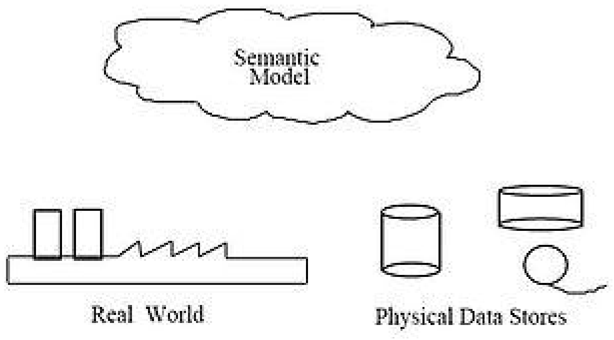
The logical data structure of a database management system (DBMS), whether hierarchical, network, or relational, cannot totally satisfy the requirements for a conceptual definition of data, because it is limited in scope and biased toward the implementation strategy employed by the DBMS. Therefore, the need to define data from a conceptual view has led to the development of semantic data modeling techniques. That is, techniques to define the meaning of data within the context of its interrelationships with other data. As illustrated in the figure. The real world, in terms of resources, ideas, events, etc., are symbolically defined within physical data stores. A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world.

DBMS의 논리적 데이터 구조는 그것이 계층적이건, 네트워크적이건, 관계형이건, 데이터

의 개념적 정의에 필요한 모든 것을 만족시킬 수는 없다. 왜냐하면 그것은 범위가 제한되어 있고 그 DBMS에서 채택하고 있는 실행 전략에 편향되어 있기 때문이다. 그러므로 개념적 견해로부터 데이터를 정의하려는 필요는 씨멘틱 데이터 모델링 기법의 발전을 도모하였다. 즉, 다른 데이터와의 상호연관성을 고려하여 데이터의 의미를 정의하는 기법이다. 아래 그림에서처럼, 자원, 아이디어, 이벤트 등과 관련된 실세계는 상징적으로 물리적 데이터 저장고로 정의되었다. 씨멘틱 데이터 모델은 그 저장된 심벌이 어떻게 실세계와 관련되어 있는지를 정의하는 추상적 개념이다. 그러므로 그 모델은 실세계를 진솔하게 표현해야만 한다.

According to Klas and Schrefl (1995), the "overall goal of semantic data models is to capture more meaning of data by integrating relational concepts with more powerful abstraction concepts known from the Artificial Intelligence field. The idea is to provide high level modeling primitives as integral part of a data model in order to facilitate the representation of real world situations".

Klas와 Schrefl(1996)에 따라, “씨멘틱 데이터 모델의 전반적인 목표는 관계적 개념과 더불어 인공지능분야에서부터 알려진 보다 강력한 추상 개념을 통합함으로써 데이터의 더 많은 의미를 수집하는 것이다. 이 아이디어는 실세계의 상황을 용이하게 표현하도록 데이터 모델의 필수적인 부분으로 고차원의 모델링 기본요소를 제공하고 있다.”



XML database

An XML database is a data persistence software system that allows data to be stored in XML format. These data can then be queried, exported and serialized into the desired format. XML databases are usually associated with document-oriented databases.

XML 데이터베이스는 데이터를 XML 포맷에 저장할 수 있는 data persistence software system 이다. 이렇게 저장된 데이터는 원하는 포맷으로 queried, exported, 그리고 serialized 될 수 있다. XML 데이터베이스는 대체로 도큐멘트 지향적 데이터베이스와 결합하고 있다.

1) In computing, a persistent data structure is a data structure that always preserves the previous version of itself when it is modified. Such data structures are effectively immutable, as their operations do not (visibly) update the structure in-place, but instead always yield a new updated structure. (A persistent data structure is not a data structure committed to persistent storage, such as a disk; this is a different and unrelated sense of the word "persistent.")

A data structure is partially persistent if all versions can be accessed but only the newest version can be modified. The data structure is fully persistent if every version can be both accessed and modified. If there is also a meld or merge operation that can create a new version from two previous versions, the data structure is called confluently persistent. Structures that are not persistent are called ephemeral.

Two major classes of XML database exist:

크게 두 가지 유형의 XML 데이터베이스가 존재 한다:

1. XML-enabled: these may either map XML to traditional database structures (such as a relational database), accepting XML as input and rendering XML as output, or more recently support native XML types within the traditional database. This term implies that the database processes the XML itself (as opposed to relying on middleware).

XML-enabled: 이러한 종류는 입력으로 XML을 받거나 출력으로 XML을 보내는 전통적인 데이터베이스 구조(관계형 데이터베이스처럼)에 XML을 위치시키도록 할 있거나 또는 보다 최근에는 그 같은 전통적인 데이터베이스 내에서 native XML 유형을 지원할 수도 있다. 이 용어가 의미하는 것은 그러한 데이터베이스는 미들웨어에 의존하는 것과는 반대로 XML 그 자체를 처리한다는 것이다.

1) Middleware is computer software that provides services to software applications beyond those available from the operating system. It can be described as "software glue". Middleware makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application.

2. Native XML (NXD): the internal model of such databases depends on XML and uses XML documents as the fundamental unit of storage, which are, however, not necessarily stored in the form of text files.

Native XML(NXD): 이러한 데이터베이스의 내부 모델은 XML에 의존하며 기본적인 저장

단위로 XML 도큐멘트를 사용하지만, 반드시 텍스트 파일의 포맷으로 저장하지는 않는다.

1) Rationale for XML in databases

O'Connell gives one reason for the use of XML in databases: the increasingly common use of XML for data transport, which has meant that "data is extracted from databases and put into XML documents and vice-versa". It may prove more efficient (in terms of conversion costs) and easier to store the data in XML format. In content-based applications, the ability of the native XML database also minimizes the need for extraction or entry of metadata to support searching and navigation. In a native XML environment, the entire content store becomes metadata through query languages such as XPath and XQuery, including content, attributes and relationships within the XML (find string "XABr" within element <para> containing attribute 123 having value "P" or "Q", only within parent Y and siblings F or G.) While this level of search capability is possible in external metadata, it requires more complex and difficult processing to reproduce the content tree in metadata.

2) XML Enabled databases

XML enabled databases typically offer one or more of the following approaches to storing XML within the traditional relational structure:

1. XML is stored into a CLOB (Character large object)
2. XML is `shredded` into a series of Tables based on a Schema [4]
3. XML is stored into a native XML Type as defined by the ISO[5]

RDBMS that support the ISO XML Type are:

1. IBM DB2 (pureXML)
2. Microsoft SQL Server
3. Oracle Database
4. PostgreSQL

Typically an XML enabled database is best suited where the majority of data are non-XML, for datasets where the majority of data are XML a Native XML Database is better suited.

< Example of XML Type Query in IBM DB2 SQL >

```
SELECT
   id, vol, xmlquery('$j/name', passing journal AS "j") AS name
FROM
```

```
   journals
WHERE
   xmlexists('$j[publisher="Elsevier"]', passing journal AS "j")
```

3) Native XML databases

The term "native XML database" (NXD) can lead to confusion. Many NXDs do not function as standalone databases at all, and do not really store the native (text) form. The formal definition from the XML: DB initiative (which appears to be inactive since 2003 states that a native XML database:

Defines a (logical) model for an XML document — as opposed to the data in that document — and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models include the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.

Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage. Need not have any particular underlying physical storage model. For example, NXDs can use relational, hierarchical, or object-oriented database structures, or use a proprietary storage format (such as indexed, compressed files). Additionally, many XML databases provide a logical model of grouping documents, called "collections". Databases can set up and manage many collections at one time. In some implementations, a hierarchy of collections can exist, much in the same way that an operating system's directory-structure works.

All XML databases now support at least one form of querying syntax. Minimally, just about all of them support XPath for performing queries against documents or collections of documents. XPath provides a simple pathing system that allows users to identify nodes that match a particular set of criteria. In addition to XPath, many XML databases support XSLT as a method of transforming documents or query-results retrieved from the database. XSLT provides a declarative language written using an XML grammar. It aims to define a set of XPath filters that can transform documents (in part or in whole) into other formats including plain text, XML, or HTML.

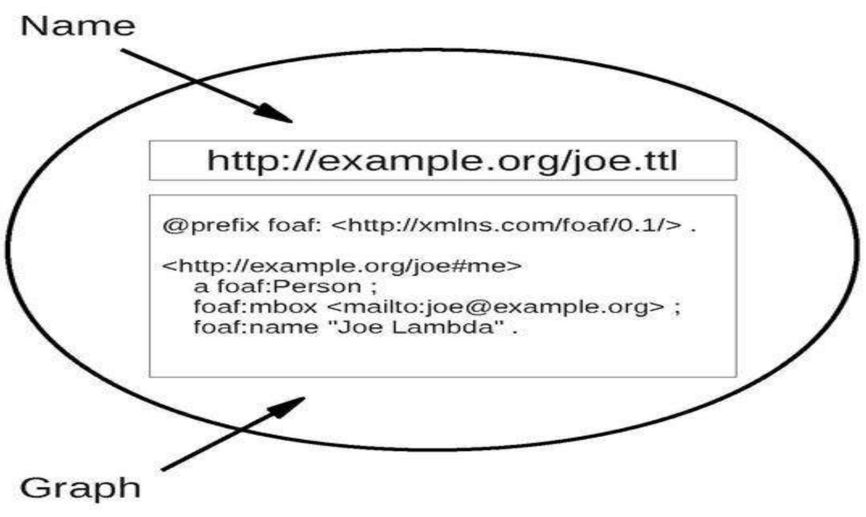
Many XML databases also support XQuery to perform querying. XQuery includes XPath as a node-selection method, but extends XPath to provide transformational capabilities. Users sometimes refer to its syntax as "FLWOR" (pronounced 'Flower') because the query may include the following clauses: 'for', 'let', 'where', 'order by' and 'return'. Traditional RDBMS vendors (who traditionally had SQL-only engines), are now shipping with hybrid SQL and XQuery engines. Hybrid SQL/XQuery engines help to query XML data alongside the relational data, in the same query expression. This approach helps in combining relational and XML data.

Most XML Databases support a common vendor neutral API called the XQuery API for Java (XQJ). The XQJ API was developed at the JCP as a standard interface to an XML/XQuery data source, enabling a Java developer to submit queries conforming to the World Wide Web Consortium (W3C) XQuery 1.0 specification and to process the results of such queries. Ultimately the XQJ API is to XML Databases and XQuery as the JDBC API is to Relational Databases and SQL

Named graph

Named graphs are a key concept of Semantic Web architecture in which a set of Resource Description Framework statements (a graph) are identified using a URI, allowing descriptions to be made of that set of statements such as context, provenance information or other such metadata. Named graphs are a simple extension of the RDF data model through which graphs can be created but the model lacks an effective means of distinguishing between them once published on the Web at large.

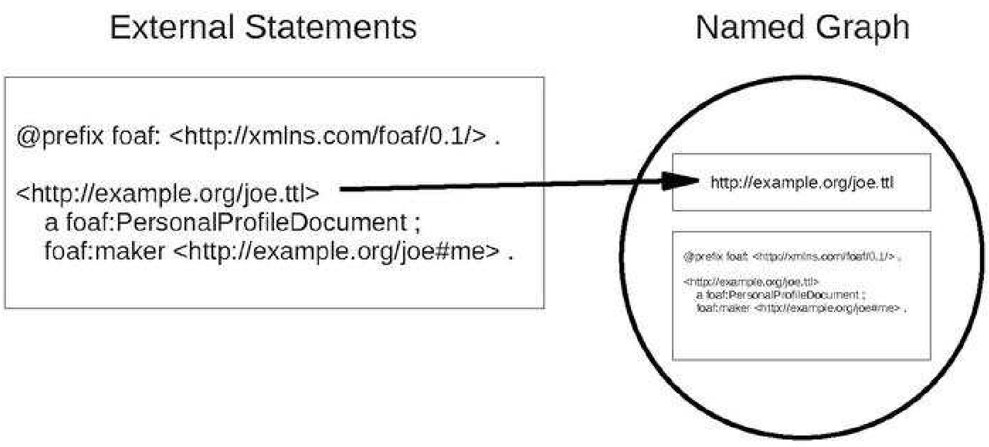
Named graphs는 한 세트의 Resource Description Framework statements(a graph)가 context, provenance(기원, 출처) 또는 메타데이터와 같은 statements의 세트를 만드는 description을 가능하게 하는 URI를 사용하는 것을 규명하는 Semantic Web 구조의 한 가지 중요한 개념이다. Named graphs는 graph를 제작할 수 있는 RDF 데이터 모델을 간단하게 확장시킨 것이지만 이 모델은 일반적으로 일단 출판되면 그것들을 구별하는 효과적 수단으로는 충분치 않다.



```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://example.org/joe#me>
   a foaf:Person ;
   foaf:mbox <mailto:joe@example.org> ;
   foaf:name "Joe Lambda" .
```

Named graphs and HTTP

One conceptualization of the Web is as a graph of document nodes identified with URIs and connected by hyperlink arcs which are expressed within the HTML documents. By doing a HTTP GET on a URI (usually via a Web browser), a somehow-related document may be retrieved. This "follow your nose" approach also applies to RDF documents on the Web in the form of Linked Data, where typically an RDF syntax is used to express data as a series of statements, and URIs within the RDF point to other resources. This Web of data has been described by Tim Berners-Lee as the "Giant Global Graph".



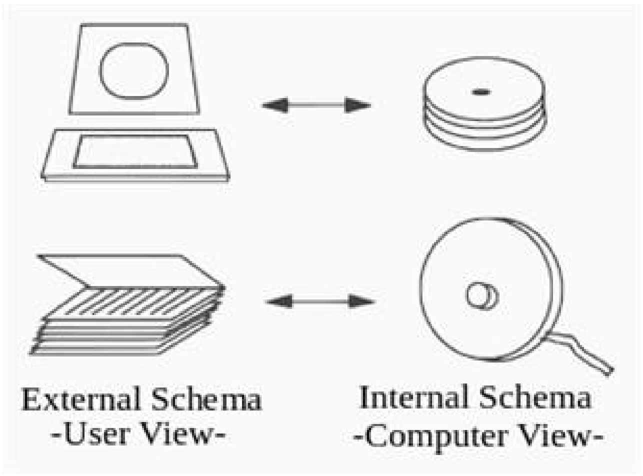
```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://example.org/joe.ttl>
   a foaf:PersonalProfileDocument ;
   foaf:maker <http://example.org/joe#me> .
```

Describing a named graph

Named graphs are a formalization of the intuitive idea that the contents of an RDF document (a graph) on the Web can be considered to be named by the URI of the document. This considerably simplifies techniques for managing chains of provenance for pieces of data and enabling fine-grained access control to the source data. Additionally trust can be managed through the publisher applying a digital signature to the data in the named graph. (Support for these facilities was originally intended to come from RDF reification, however that approach proved problematic.

Named graphs는 웹의 RDF 도큐멘트(a graph)의 콘텐트가 그 도큐멘트의 URK에 의해 명명되어진다고 여겨질 수 있다는 직관적 아이디어를 나타내는 것이다. 이것은 상당히 여러 가지 데이터의 출처 고리를 관리하고 그 원천 데이터의 접근을 효과적으로 통제할 수 있는 기술을 단순화시킨 것이다. 추가적으로 그 같은 named graph에 있는 데이터에 디지털 서명를 적용하는 출판사를 통하여 신뢰가 관리될 수도 있다.(이러한 기능의 지원은 원리 RDF reification(구상화) 단계에서부터 하려고 했으나 그러한 시도가 문제가 있는 것으로 파악되었다).

6.2 External, conceptual, and internal views



Traditional view of data

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

데이터베이스관리시스템은 데이터베이스 데이터에 대하여 3가지의 견해를 제공한다:

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.

외적 차원은 엔드유저의 각 그룹의 데이터베이스에 있는 데이터의 조직을 어떻게 아는가를 정의한다. 단일 데이터베이스는 외적 차원에서 어느 정도의 견해를 가질 수 있다.

The conceptual 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.

개념적 차원은 다양한 외적 견해를 호환 가능한 보편적 견해로 통합한다. 이것은 오든 외적 견해의 통합을 제공한다. 이것은 다양한 데이터베이스 엔드유저의 범위를 벗어나서 그것보다는 데이터베이스 개발자와 데이터베이스 행정가에 대한 관심의 대상이다.

The internal level (or physical level) is the internal organization of data inside a DBMS (see Implementation section below). 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.

내적 차원(또는 물리적 차원)은 DBMS에 있는 데이터의 내적 조직을 말한다. 이것은 비용, 성능, scalability, 그리고 기타 운영업무와 관련이 있다. 이것은 성능을 높이기 위하여 색인과 같은 저장 구조를 사용함으로써 데이터의 저장 외관을 다룬다. 경우에 따라서, 만일 성능 확인이 그 같은 잉여정보용으로 존재한다면, 이것은 종속적 데이터로부터 계산된 개별적 뷰(물질화된 견해들)의 데이터를 저장한다. 이것은 모든 활동에 대하여 전체적인 성능을 최적화하려는 시도에 따라 가능한 한 부딪치면서 모든 외적 뷰의 성능 요구조건과 균형을 맞춘다.

While there is typically only one conceptual (or logical) and physical (or 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 the interest of the human resources department. Thus different departments need different views of the company's database.

전형적으로 데이터에 대한 단지 한가지의 개념적(또는 논리적) 그리고 물리적(또는 내적) 뷰가 존재한다 하더라도, 어는 정도의 서로 다른 외적 뷰가 존재한다. 이것은 이용자로 하여금 기술적, 처리적인 입장보다는 보다 다양한 업무-관련 방법으로 데이터베이스 정보를 파악할 수 있도록 한다. 예를 들어, 회사의 재무부는 회사 경비의 일부로서 모든 직원에 대한 급료명세서를 필요로 한다. 그러나 인사부의 관심인 직원에 대한 상세정보는 필요하지 않다. 그러므로 서로 다른 부서들은 그 회사 데이터베이스에서 서로 다른 뷰를 필요로 한다.

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.

3가지 차원의 데이터베이스 구조는 관계형 모델의 중요한 추진력의 하나인 데이터의 독립성과 관련이 있다. 어느 수준에서는 변화가 일어난다는 그 같은 아이디어는 보다 고차원에 있는 뷰에는 영향을 끼치지 않는다. 예를 들어, 내적 차원에서의 변화는 개념적 차원의 인터페이스를 사용하기 위하여 작성된 응용 프로그램에 영향을 끼치지 않는다. 성능을 향상시키기 위하여 물리적 변화에 따른 영향을 줄인다.

The conceptual view provides a level of indirection between internal and external. On 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 is 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 (see Implementation section below), requires a different level of detail and uses its own types of data structure types.

개념적 뷰는 내적 그리고 외적 간의 불간섭 수준을 제공한다. 한편으로 이것은 서로 다른 외적 뷰의 구조와는 독립된 데이터베이스에 대한 일반 뷰를 제공한다. 또 한편으로 이것은 데이터가 어떻게 저장되거나 관리되는지에 대한 상세한 내용을 밝힌다(내적 수준). 원칙적으로 모든 차원 그리고 심지어 모든 외적 뷰는 하나의 다양한 데이터 모델로 표현될 수 있다. 실재적으로, 대체로 특정한 DBMS는 외적 그리고 개념적 차원용으로 동일한 데이터 모델을 사용하고 있다(예를 들어 관계형 모델). DBMS 안에 숨어있는 그리고 그것의 실행에 의존하는 내적 차원은 서로 다른 차원의 상세한 내용을 필요로 하며 그것 자체의 유형인 데이터 구조 유형을 사용한다.

Separating the external, conceptual and internal levels was a major feature of the relational database model implementations that dominate 21st century databases.

외적인 것과는 별도로, 개념적 그리고 내적 차원은 21세기 데이터메이스를 지배하고 있는 관계형 데이터베이스 모델의 실행에 있는 중요한 특징이다.

7. Database languages

Database languages are special-purpose languages, which do one or more of the following:

데이터베이스 언어는 특별한 목적의 언어이며 다음과 같거나 더 많다:

Data definition language - defines data types and the relationships among them

데이터정의어 - 데이터 유형과 그것들 간의 관계를 정의한다.

Data manipulation language - performs tasks such as inserting, updating, or deleting data occurrences

데이터 조작어 - 데이터의 입력, 갱신, 또는 삭제와 같은 업무를 수행한다.

Query language - allows searching for information and computing derived information

쿼리어 - 정보의 탐색과 유도된 정보를 계산하도록 한다.

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

데이터베이스 언어는 특별한 데이터 모델에 한정적이다. 잘 알려진 예는 다음과 같다:

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 Standards(ISO) in 1987. The standards have been regularly enhanced since and is supported (with varying degrees of conformance) by all mainstream commercial relational DBMSs.

SQL은 단일 언어로 데이터 정의, 데이터 조작, 그리고 쿼리의 역할을 결합하고 있다. 이것은 비록 어떤 점에서는 Codd가 설명한 관계형 모델(예를 들어 테이블의 컬럼과 로우는 순서에 맞춰질 수 있다)과는 동떨어져 있다하더라도 관계형 모델의 첫 번째 상업적 언어들 중의 하나이다. SQL은 1986년에 ANSI의 표준이 되었으며, 1987년에 ISO의 표준이 되었다. 이 표준들은 그 이후로 정기적으로 갱신되고 있으며 모든 주요 상업적 관계형 DBMSs에 의해 적합도가 서로 다르지만 지원을 받고 있다

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.

OQL은 Object Data Management Group에서 만든 객체모델 언어 표준이다. 이것은 JDOQL과 EJB QL과 같은 보다 새로운 쿼리 언어의 디자인에 영향을 끼치고 있다.

1) Object Query Language (OQL) is a query language standard for object-oriented databases modeled after SQL. OQL was developed by the Object Data Management Group (ODMG). Because of its overall complexity no vendor has ever fully implemented the complete OQL. OQL has influenced the design of some of the newer query languages like JDOQL and EJB QL, but they can't be considered as different flavors of OQL.

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.

XQuery는 MarkLogic과 eXist와 같은 XML 데이터베이스 시스템에서, 그리고 Oracle과 DB2와 같은 XML 기능을 갖춘 관계형 데이터베이스에서, 그리고 또한 Saxon과 같은 메모리에 내장된 XML 프로세서에서 사용하고 있는 표준 XML 쿼리 언어이다

SQL/XML combines XQuery with SQL.

SQL/XML은 XQery와 SQL이 결합된 것이다.

A database language may also incorporate features like:

데이터베이스 언어는 또한 다음과 같은 특징을 가질 수도 있다:

DBMS-specific Configuration and storage engine management

DBMS에 한정된 맞춤구조와 저장엔진 관리

Computations to modify query results, like counting, summing, averaging, sorting, grouping, and cross-referencing

계산, 합계, 평균, 분류, 집단화, 그리고 상호참조와 같은 쿼리 결과의 변경을 위한 계산

Constraint enforcement (e.g. in an automotive database, only allowing one engine type per car)

제한점(예, 자동차 데이터베이스에서 단지 차 당 한 종류의 엔진만 허용)

Application programming interface version of the query language, for programmer convenience

프로그래머의 편의를 위하여 쿼리 언어의 응용 프로그래밍 인터페이스 버전

8. Performance, security, and availability

Because of the critical importance of database technology to the smooth running of an enterprise, database systems include complex mechanisms to deliver the required performance, security, and availability, and allow database administrators to control the use of these features.

기업의 원활한 운영을 위하여 데이터베이스 기술이 매우 중요하기 때문에, 데이터베이스 시스템에는 요구된 성능, 보안, 그리고 이용가능성을 제공할 수 있는 복잡한 메커니즘이 포함되어 있으며, 데이터베이스 운영자로 하여금 이러한 특징의 사용을 제어할 수 있도록 한다.

8.1 Database 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. 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 utilizing the operating systems' file systems as intermediates for storage layout), storage properties and configuration setting 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 in 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).

데이터베이스 저장고는 데이터베이스의 물리적 실체의 콘테이너이다. 이것은 데이터베이스의 구조에 있는 내적(물리적) 차원으로 구성된다. 또한 이것은 필요할 때 그 내적 차원으로부터 개념적 차원과 외적 차원을 재구축하는데 필요한 모든 정보(예, “데이터의 데이터”인 메타데이터와 내적 데이터 구조들)를 포함하고 있다. 데이터를 항구적인 저장고에 넣은 것은 일반적으로 그 데이터베이스 엔진 a.k.a.(also known as: 별칭은, 별명은) “저장고 엔진”의 책임이다. 기초가 된 운영체제를 통하여 DBMS에 접근하는 것이 전형적이라 하더라도(그리고 종종 저장고 레이아우트를 위한 매개체로서 운영체제의 파일 시스템을 활성화시키더라도), 저장고의 성질과 구성은 그 DBMS의 효율적인 운영을 위하여 극히 중요하며, 따라서서 데이터베이스 행정가에 의해 밀접하게 관리되고 있다. DBMS, 현재 운영 중인 DBMS는 여러 유형의 저장고(예, 메모리와 외적 저장고)가 딸려있는 자신의 데이터베이스를 늘 가지고 있다. 데이터베이스 데이터와 추가적으로 필요한 정보가 아마도 매우 많은데 이것들은 bits로 암호화 되어있다. 전형적으로 데이터는 개념적 그리고 외적 차원의 데이터를 보는 방식과는 완전히 다르게

보이는 구조 속의 저장고에 들어 있다. 그러나 그 데이터로부터 추가적으로 필요한 유형의 정보를 계산하는 것(예, 데이터베이스에 쿼리가 발생할 때)뿐만 아니라 이용자와 프로그램에 의해 필요할 때, 이러한 차원들의 재구성을 최적화시킬 수 있는 방식으로 들어 있다.

1) Metadata is "data about data". The term is ambiguous, as it is used for two fundamentally different concepts (types). Structural metadata is about the design and specification of data structures and is more properly called "data about the containers of data"; descriptive metadata, on the other hand, is about individual instances of application data, the data content.

Metadata are traditionally found in the card catalogs of libraries. As information has become increasingly digital, metadata are also used to describe digital data using metadata standards specific to a particular discipline. By describing the contents and context of data files, the quality of the original data/files is greatly increased. For example, a webpage may include metadata specifying what language it is written in, what tools were used to create it, and where to go for more on the subject, allowing browsers to automatically improve the experience of users.

Libraries
Metadata have been used in various forms as a means of cataloging archived information. The Dewey Decimal System employed by libraries for the classification of library materials is an early example of metadata usage. Library catalogues used 3x5 inch cards to display a book's title, author, subject matter, and a brief plot synopsis along with an abbreviated alpha-numeric identification system which indicated the physical location of the book within the library's shelves. Such data help classify, aggregate, identify, and locate a particular book. Another form of older metadata collection is the use by US Census Bureau of what is known as the "Long Form." The Long Form asks questions that are used to create demographic data to find patterns of distribution.

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

어떤 DBMS는 특정한 문자암호화를 사용하여 데이터를 저장하도록 하므로, 다수의 암호화가 동일한 데이터베이스에 사용될 수 있다.

1) A character encoding system consists of a code that pairs each character from a given repertoire with something else—such as a bit pattern, sequence of natural numbers, octets, or electrical pulses—in order to facilitate the transmission of data (generally numbers or text) through telecommunication networks or for

data storage. Other terms such as character set, character map, codeset, and code page are used almost interchangeably, but these terms have related but distinct meanings described below.

Early character codes associated with the optical or electrical telegraph could only represent a subset of the characters used in written language, sometimes restricted to upper case letters, numerals and some punctuation only. The low cost of digital representation of data in modern computer systems allows more elaborate character codes (such as Unicode) which represent more of the characters used in many written languages. Character encoding using internationally accepted standards permits worldwide interchange of text in electronic form.

History

Early binary repertoires include Bacon's cipher, Braille, International maritime signal flags, and the 4-digit encoding of Chinese characters for a Chinese telegraph code (Hans Schjellerup, 1869). Common examples of character encoding systems include Morse code, the Baudot code, the American Standard Code for Information Interchange (ASCII) and Unicode.

Morse code was introduced in the 1840s and is used to encode each letter of the Latin alphabet, each Arabic numeral, and some other characters via a series of long and short presses of a telegraph key. Representations of characters encoded using Morse code varied in length.

The Baudot code, a 5-bit encoding, was created by Émile Baudot in 1870, patented in 1874, modified by Donald Murray in 1901, and standardized by CCITT as International Telegraph Alphabet No. 2 (ITA2) in 1930.

ASCII was introduced in 1963 and is a 7-bit encoding scheme used to encode letters, numerals, symbols, and device control codes as fixed-length codes using integers.

IBM's Extended Binary Coded Decimal Interchange Code (usually abbreviated EBCDIC) is an 8-bit encoding scheme developed in 1963.

The limitations of such sets soon became apparent, and a number of ad hoc methods were developed to extend them. The need to support more writing systems for different languages, including the CJK family of East Asian scripts, required support for a far larger number of characters and demanded a systematic approach to character encoding rather than the previous ad hoc approaches.

Code unit

A code unit is a bit sequence used to encode the characters of a repertoire.

With US-ASCII, code unit is 7 bits.
With UTF-8, code unit is 8 bits.
With EBCDIC, code unit is 8 bits.
With UTF-16, code unit is 16 bits.
With UTF-32, code unit is 32 bits.

Encodings associate their meaning with either a single code unit value or a sequence of code units as one value.

Unicode encoding model

Unicode and its parallel standard, the ISO/IEC 10646 Universal Character Set, together constitute a modern, unified character encoding. Rather than mapping characters directly to octets (bytes), they separately define what characters are available, their numbering, how those numbers are encoded as a series of "code units" (limited-size numbers), and finally how those units are encoded as a stream of octets. The idea behind this decomposition is to establish a universal set of characters that can be encoded in a variety of ways.[1] To describe this model correctly one needs more precise terms than "character set" and "character encoding." The terms used in the modern model follow:

A character repertoire is the full set of abstract characters that a system supports. The repertoire may be closed, i.e. no additions are allowed without creating a new standard (as is the case with ASCII and most of the ISO-8859 series), or it may be open, allowing additions (as is the case with Unicode and to a limited extent the Windows code pages). The characters in a given repertoire reflect decisions that have been made about how to divide writing systems into basic information units. The basic variants of the Latin, Greek, and Cyrillic alphabets, can be broken down into letters, digits, punctuation, and a few special characters like the space,[citation needed] which can all be arranged in simple linear sequences that are displayed in the same order they are read. Even with these alphabets, however, diacritics pose a complication: they can be regarded either as part of a single character containing a letter and diacritic (known as a precomposed character), or as separate characters. The former allows a far simpler text handling system but the latter allows any letter/diacritic combination to be used in text. Ligatures pose similar problems. Other writing systems, such as Arabic and Hebrew, are represented with more complex character repertoires due to the need to accommodate things like bidirectional text and glyphs that are joined together in different ways for different situations.

A coded character set (CCS) specifies how to represent a repertoire of characters using a number of (typically non-negative) integer values called code points. For example, in a given repertoire, a character representing the capital letter "A" in the Latin alphabet might be assigned to the integer 65, the character for "B" to 66, and so on. A complete set of characters and corresponding integers is a coded character set. Multiple coded character sets may share the same repertoire; for example ISO/IEC 8859-1 and IBM code pages 037 and 500 all cover the same repertoire but map them to different codes. In a coded character set, each code point only represents one character, i.e., a coded character set is a function.

A character encoding form (CEF) specifies the conversion of a coded character set's integer codes into a set of limited-size integer code values that facilitate storage in a system that represents numbers in binary form using a fixed number of bits (i.e. practically any computer system). For example, a system that stores numeric information in 16-bit units would only be able to directly represent integers from 0 to 65,535 in each unit, but larger integers could be represented if more than one 16-bit unit could be used. This is what a CEF accommodates: it defines a way of mapping a single code point from a range of, say, 0 to 1.4 million, to a series of one or more code values from a range of, say, 0 to 65,535.

The simplest CEF system is simply to choose large enough units that the values from the coded character set can be encoded directly (one code point to one code value). This works well for coded character sets that fit in 8 bits (as most legacy non-CJK encodings do) and reasonably well for coded character sets that fit in 16 bits (such as early versions of Unicode). However, as the size of the coded character set increases (e.g. modern Unicode requires at least 21 bits/character), this becomes less and less efficient, and it is difficult to adapt existing systems to use larger code values. Therefore, most systems working with later versions of Unicode use either UTF-8, which maps Unicode code points to variable-length sequences of octets, or UTF-16, which maps Unicode code points to variable-length sequences of 16-bit words.

Next, a character encoding scheme (CES) specifies how the fixed-size integer code values should be mapped into an octet sequence suitable for saving on an octet-based file system or transmitting over an octet-based network. With Unicode, a simple character encoding scheme is used in most cases, simply specifying whether the bytes for each integer should be in big-endian or little-endian order (even this isn't needed with UTF-8). However, there are also compound character encoding schemes, which use escape sequences to switch between several simple schemes (such as ISO/IEC 2022), and compressing schemes, which try to minimise the number of bytes used per code unit (such as SCSU, BOCU, and Punycode). See comparison of Unicode encodings for a detailed discussion.

Finally, there may be a higher level protocol which supplies additional information that can be used to select the particular variant of a Unicode character, particularly where there are regional variants that have been 'unified' in Unicode as the same character. An example is the XML attribute xml:lang.

The Unicode model reserves the term character map for historical systems which directly assign a sequence of characters to a sequence of bytes, covering all of CCS, CEF and CES layers.

Character sets, code pages, and character maps

In computer science, the terms character encoding, character map, character set or code page were historically synonymous, as the same standard would specify a repertoire of characters and how they were to be encoded into a stream of code units – usually with a single character per code unit. The terms now have related but distinct meanings, reflecting the efforts of standards bodies to use precise terminology when writing about and unifying many different encoding systems. Regardless, the terms are still used interchangeably, with character set being nearly ubiquitous.

A code page usually means a byte oriented encoding, but with regard to some suite of encodings (covering different scripts), where many characters share the same codes in most or all those code pages. Well known code page suites are "Windows" (based on Windows-1252) and "IBM"/"DOS" (based on code page 437), see Windows code page for details. Most, but not all, encodings referred to as code pages are single-byte encodings (but see octet on byte size.)

IBM's Character Data Representation Architecture (CDRA) designates with coded character set identifiers (CCSIDs) and each of which is variously called a charset, character set, code page, or CHARMAP.

The term code page does not occur in Unix or Linux where charmap is preferred, usually in the larger context of locales.

Contrasted to CCS above, a character encoding is a map from abstract characters to code words. A character set in HTTP (and MIME) parlance is the same as a character encoding (but not the same as CCS).

Legacy encoding is a term sometimes used to characterize old character encodings, but with an ambiguity of sense. Most of its use is in the context of Unicodification, where it refers to encodings that fail to cover all Unicode code points, or, more generally, using a somewhat different character repertoire: several code points representing one Unicode character, or versa (see e.g. code page 437). Some sources refer to an encoding as legacy only because it preceded Unicode. All Windows code pages are usually referred to as legacy, both because they antedate Unicode and because they are unable to represent all 221 possible Unicode code points.

Character encoding translation

As a result of having many character encoding methods in use (and the need for backward compatibility with archived data), many computer programs have been developed to translate data between encoding schemes. Some of these are cited below.

Cross-platform:

Web browsers –most modern web browsers feature automatic character encoding detection. On Firefox 3, for example, see the View/Character Encoding submenu.

iconv –program and standardized API to convert encodings luit – program that converts encoding of input and output to programs running interactively

convert_encoding.py –Python based utility to convert text files between arbitrary encodings and line endings.

decodeh.py –algorithm and module to heuristically guess the encoding of a string.

International Components for Unicode –A set of C and Java libraries to perform charset conversion. uconv can be used from ICU4C.

chardet –This is a translation of the Mozilla automatic-encoding-detection code into the Python computer language.

The newer versions of the Unix file command attempt to do a basic detection of character encoding (also available on Cygwin).

Unix-like:

cmv – simple tool for transcoding filenames.
convmv – convert a filename from one encoding to another.
cstocs – convert file contents from one encoding to another for the Czech and Slovak languages.

enca - analyzes encodings for given text files.
recode - convert file contents from one encoding to another
utrac - convert file contents from one encoding to another.

Windows:

Encoding.Convert - .NET API
MultiByteToWideChar/WideCharToMultiByte - Convert from ANSI to Unicode & Unicode

to ANSI
cscvt - character set conversion tool
enca - analyzes encodings for given text files.

Common character encodings
*ISO 646
  ASCII
*EBCDIC
  CP37
  CP930
  CP1047
*ISO 8859:
  ISO 8859-1 Western Europe
  ISO 8859-2 Western and Central Europe
  ISO 8859-3 Western Europe and South European (Turkish, Maltese plus Esperanto)
  ISO 8859-4 Western Europe and Baltic countries (Lithuania, Estonia, Latvia and Lapp)
  ISO 8859-5 Cyrillic alphabet
  ISO 8859-6 Arabic
  ISO 8859-7 Greek
  ISO 8859-8 Hebrew
  ISO 8859-9 Western Europe with amended Turkish character set
  ISO 8859-10 Western Europe with rationalised character set for Nordic languages,
    including complete Icelandic set
  ISO 8859-11 Thai
  ISO 8859-13 Baltic languages plus Polish
  ISO 8859-14 Celtic languages (Irish Gaelic, Scottish, Welsh)
  ISO 8859-15 Added the Euro sign and other rationalisations to ISO 8859-1
  ISO 8859-16 Central, Eastern and Southern European languages (Albanian,

Croatian,
Hungarian, Polish, Romanian, Serbian and Slovenian, but also French,
German, Italian and Irish Gaelic)
*CP437, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872
*MS-Windows character sets:
Windows-1250 for Central European languages that use Latin script, (Polish, Czech,
Slovak, Hungarian, Slovene, Serbian, Croatian, Romanian and Albanian)
Windows-1251 for Cyrillic alphabets
Windows-1252 for Western languages
Windows-1253 for Greek
Windows-1254 for Turkish
Windows-1255 for Hebrew
Windows-1256 for Arabic
Windows-1257 for Baltic languages
Windows-1258 for Vietnamese
*Mac OS Roman
*KOI8-R, KOI8-U, KOI7
*MIK
*ISCII
*TSCII
*VISCII
*JIS X 0208 is a widely deployed standard for Japanese character encoding that has
several encoding forms.
Shift JIS (Microsoft Code page 932 is a dialect of Shift_JIS)
EUC-JP
ISO-2022-JP
*JIS X 0213 is an extended version of JIS X 0208.
Shift_JIS-2004
EUC-JIS-2004
ISO-2022-JP-2004
*Chinese Guobiao
GB 2312
GBK (Microsoft Code page 936)
GB 18030
*Taiwan Big5 (a more famous variant is Microsoft Code page 950)
*Hong Kong HKSCS

*Korean

KS X 1001 is a Korean double-byte character encoding standard

EUC-KR

ISO-2022-KR

*Unicode (and subsets thereof, such as the 16-bit 'Basic Multilingual Plane').

See UTF-8

*ANSEL or ISO/IEC 6937

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.

여러 가지 저급의 데이터베이스 저장 구조가 그 데이터 모델을 연속적으로 나열하기 위하여 저장 엔진에 의해 사용됨으로써 그것들이 선택된 매체에 기록될 수 있다. 색인과 같은 기법이 성능개선을 위해 사용될 수 있다. 전통적인 저장고는 열-지향적이만 행-지향적이면서 상호관계 데이터베이스로 존재한다.

1) A correlation database is a database management system (DBMS) that is data-model-independent and designed to efficiently handle unplanned, ad hoc queries in an analytical system environment. It was developed in 2005 by database architect Joseph Foley.

Unlike relational database management systems, which use a records-based storage approach, or column-oriented databases which use a column-based storage method, a correlation database uses a value-based storage (VBS) architecture in which each unique data value is stored only once and an auto-generated indexing system maintains the context for all values

8.1.1 Database 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 of 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.

종종 저장의 잉여는 성능을 높이기 위하여 채택된다. 하나의 일반적인 예가 가끔 요구된 외적 뷰나 쿼리 결과로 이루어지는 실현 뷰를 저장하는 것이다. 그러한 뷰를 저장하는 것은

그것들이 필요할 때마다 계산을 해야 하는 비용을 절약한다. 실현 뷰의 규모를 축소하는 것은 원래의 갱신된 데이터베이스 데이터와 그것들을 동기화시키도록 갱신할 때 그리고 저장 잉여의 비용이라는 과징금을 부과하는 것이다.

1) A materialized view is a database object that contains the results of a query. For example, it may be a local copy of data located remotely, or may be a subset of the rows and/or columns of a table or join result, or may be a summary based on aggregations of a table's data. Materialized views, which store data based on remote tables, are also known as snapshots. A snapshot can be redefined as a materialized view.

8.1.2 Database and database object 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 a 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.

경우에 따라서, 데이터베이스는 데이터의 이용가능성(동일한 데이터베이스 객체에 동시에 다수의 최종이용자가 접근할 수 있는 성능을 개선시키고, 분산 데이터베이스의 부분적 잘못이 발생한 경우에 재생성)을 높이기 위하여 데이터베이스 객체 복제(하나이상의 사본을 가지고)를 사용하여 저장 잉여를 사용한다. 복제된 객체를 갱신하는 것은 그 객체의 사본간에 동시에 이루어져야 한다. 많은 경우에서 그 데이터베이스 전체가 복제되고 있다.

8.2 Database 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).

데이터베이스 보안은 그 데이터베이스의 콘텐트, 소유자, 이용자를 보호하는 모든 사안을 다룬다. 이것의 범위는 의도적인 불법이용자로부터의 보호에서부터 권한이 없는 객체(예, 사람이나 컴퓨터 프로그램)에 의한 무의식적인 데이터베이스 접근까지이다.

Database access control deals with controlling who (a person or a certain computer program) is 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 utilizing 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.

데이터베이스 접근통제는 누가(사람이나 어떤 컴퓨터 프로그램) 데이터베이스에 있는 어떤 정보에 접근할 수 있는지를 통제한다. 그 정보는 특별한 데이터베이스 객체(예, 레코드 종류, 특수 레코드, 데이터 구조), 어떤 객체에 대한 어떤 계산(예, 쿼리 유형 또는 특별한 쿼리), 또는 전자로의 특별한 접근 통로의 활성화(예, 특수한 색인이나 정보접근을 위한 기타 데이터 구조의 이용)에 관한 것들일 수 있다. 데이터베이스 접근통제는 전용 보호 안전용 DBMS 인터페이스를 사용하는 특별한 권한을 받은 사람(데이터베이스 소유자)에 의해서 설정된다.

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.

이것은 개인적 차원에서 직접적으로 또는 개인의 양도와 집단에 대한 특권에 의해, 또는 대부분의 정교한 모델에서는 권리가 부여된 역할에 대하여 개인과 집단의 양도를 통해 관리될 수 있다. 데이터 보안은 불법이용자가 데이터베이스를 살펴보거나 갱신하는 것을 예방한다. 패스워드를 사용함으로써, 이용자는 데이터베이스 전체와 소위 “하위 스키마”라 부르는 부분집합에 접근할 수 있다. 예를 들어, 직원 데이터베이스는 직원 개인에 대한 모든 데이터를 포함할 수 있지만, 한 이용자 집단은 단지 급료 데이터만을 볼 수 있는 자격이 있는 반면에 또 다른 집단은 작업이력과 의료 데이터에만 접근할 수 있다. 만일 DBMS가 데이터베이스의 입력과 갱뿐만 아니라 그것에 대한 질문에 대하여 쌍방향성을 제공한다면, 이러한 기능을 통하여 개인별로 데이터베이스의 관리가 가능하다.

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).

데이터 보안은 일반적으로 특별한 규모의 데이터를 물리적(다시 말해서, 부패, 파괴, 제거로부터; 예, 물리적 보안 참조)으로 또는 그것들 전부 또는 일부를 의미 있는 정보로 해석한 것(예, 구성하고 있는 일련의 비트정보를 살펴봄으로써 유효한 신용카드 번호라고 결론짓는 것; 예, 데이터 암호화를 참조)

1) Physical security describes security measures that are designed to deny unauthorized access to facilities, equipment and resources, and to protect personnel and property from damage or harm (such as espionage, theft, or terrorist attacks). Physical security involves the use of multiple layers of interdependent systems which include CCTV surveillance, security guards, protective barriers, locks, access control protocols, and many other techniques.

Overview

Canadian Embassy in Washington, D.C. showing planters being used as vehicle barriers to increase the standoff distance, and barriers and gates along the vehicle entrancePhysical security systems for protected facilities are generally intended to:

*deter potential intruders (e.g. warning signs and perimeter markings);
*distinguish authorized from unauthorized people (e.g. using keycards/access badges)
*delay, frustrate and ideally prevent intrusion attempts (e.g. strong walls, door locks and
safes);
*detect intrusions and monitor/record intruders (e.g. intruder alarms and CCTV systems);
and
*trigger appropriate incident responses (e.g. by security guards and police).

It is up to security designers, architects and analysts to balance security controls against risks, taking into account the costs of specifying, developing, testing, implementing, using, managing, monitoring and maintaining the controls, along with broader issues such as aesthetics, human rights, health and safety, and societal norms or conventions. Physical access security measures that are appropriate for a high security prison or a military site may be inappropriate in an office, a home or a vehicle, although the principles are similar.

Elements and design

*Deterrence methods

The goal of deterrence methods is to convince potential attackers that a successful attack is unlikely due to strong defenses.

The initial layer of security for a campus, building, office, or other physical space uses crime prevention through environmental design to deter threats. Some of the most common examples are also the most basic: warning signs or window stickers, fences, vehicle barriers, vehicle height-restrictors, restricted access points, security lighting and trenches.

*Physical barriers

Spikes atop a barrier wall act as a deterrent to people trying to climb over the wallPhysical barriers such as fences, walls, and vehicle barriers act as the outermost layer of security. They serve to prevent, or at least delay, attacks, and also act as a psychological deterrent by defining the perimeter of the facility and making intrusions seem more difficult. Tall fencing, topped with barbed wire, razor wire or metal spikes are often emplaced on the perimeter of a property, generally with some type of signage that warns people not to attempt to enter. However, in some facilities imposing perimeter walls/fencing will not be possible (e.g. an urban office building that is directly adjacent to public sidewalks) or it may be aesthetically unacceptable (e.g. surrounding a shopping center with tall fences topped with razor wire); in this case, the outer security perimeter will be defined as the walls/windows/doors of the structure itself.[9]

*Natural surveillance

Another major form of deterrence that can be incorporated into the design of facilities is natural surveillance, whereby architects seek to build spaces that are more open and visible to security personnel and authorized users, so that intruders/attackers are unable to perform unauthorized activity without being seen. An example would be decreasing the amount of dense, tall vegetation in the landscaping so that attackers cannot conceal themselves within it, or placing critical resources in areas where intruders would have to cross over a wide, open space to reach them (making it likely that someone would notice them).

*Security lighting

Security lighting is another effective form of deterrence. Intruders are less likely to enter well-lit areas for fear of being seen. Doors, gates, and other entrances, in particular, should be well lit to allow close observation of people entering and exiting. When lighting the grounds of a facility, widely-distributed low-intensity lighting is generally superior to small patches of high-intensity lighting, because the latter can have a tendency to create blind spots for security personnel and CCTV cameras. It is important to place lighting in a manner that makes it difficult to tamper with (e.g. suspending lights from tall poles), and to ensure that there is a backup power supply so that security lights will not go out if the electricity is cut off.

*Intrusion detection and electronic surveillance

*Alarm systems and sensors

Alarm systems can be installed to alert security personnel when unauthorized access is attempted. Alarm systems work in tandem with physical barriers, mechanical systems, and security guards, serving to trigger a response when these other forms of security have been breached. They consist of sensors including motion sensors, contact sensors, and glass break detectors.

However, alarms are only useful if there is a prompt response when they are triggered. In the reconnaissance phase prior to an actual attack, some intruders will test the response time of security personnel to a deliberately tripped alarm system. By measuring the length of time it takes for a security team to arrive (if they arrive at all), the attacker can determine if an attack could succeed before authorities arrive to neutralize the threat. Loud audible alarms can also act as a psychological deterrent, by notifying intruders that their presence has been detected. In some jurisdictions, law enforcement will not respond to alarms from intrusion detection systems unless the activation has been verified by an eyewitness or video. Policies like this one have been created to combat the 94-99 percent rate of false alarm activation in the United States.

*Video surveillance: Closed-circuit television cameras

Surveillance cameras can be a deterrent when placed in highly visible locations, and are also useful for incident verification and historical analysis. For example, if alarms are being generated and there is a camera in place, the camera could be viewed to verify the alarms. In instances when an attack has already occurred and a camera is in place at the point of attack, the recorded video can be reviewed. Although the term closed-circuit television (CCTV) is common, it is quickly becoming outdated as more video systems lose the closed circuit for signal transmission and are instead transmitting on IP camera networks.

Video monitoring does not necessarily guarantee that a human response is made to an intrusion. A human must be monitoring the situation realtime in order to respond in a timely manner. Otherwise, video monitoring is simply a means to gather evidence to be analyzed at a later time. However, advances in information technology are reducing the amount of work required for video monitoring, through automated video analytics.

*Access control

Access control methods are used to monitor and control traffic through specific access points and areas of the secure facility. This is done using a variety of systems including CCTV surveillance, identification cards, security guards, and electronic/mechanical control systems such as locks, doors, and gates.

*Mechanical access control systems: An electronic access control system, controlling entry through a door.

Mechanical access control systems include gates, doors, and locks. Key control of the locks becomes a problem with large user populations and any user turnover. Keys quickly become unmanageable, often forcing the adoption of electronic access control.

*Electronic access control systems

Electronic access control easily manages large user populations, controlling for user lifecycles times, dates, and individual access points. For example a user's access rights could allow access from 0700h to 1900h Monday through Friday and expires in 90 days.

An additional sub-layer of mechanical/electronic access control protection is reached by integrating a key management system to manage the possession and usage of mechanical keys to locks or property within a building or campus.[citation needed]

*Identification systems and access policies

Another form of access control (procedural) includes the use of policies, processes and procedures to manage the ingress into the restricted area. An example of this is the deployment of security personnel conducting checks for authorized entry at predetermined points of entry. This form of access control is usually supplemented by the earlier forms of access control (i.e. mechanical and electronic access control), or simple devices such as physical passes.

*Security personnel

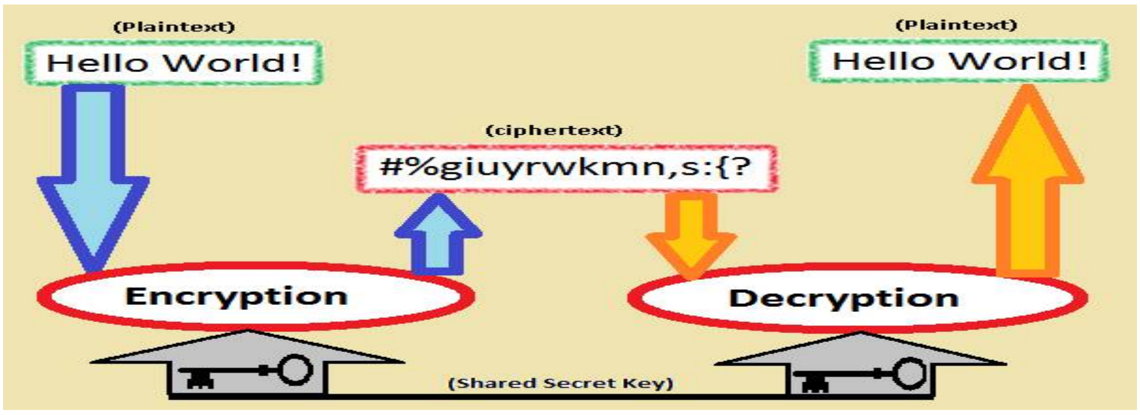
Security personnel play a central role in all layers of security. All of the technological systems that are employed to enhance physical security are useless without a security force that is trained in their use and maintenance, and which knows how to properly respond to breaches in security. Security personnel perform many functions: as patrols and at checkpoints, to administer electronic access control, to respond to alarms, and to monitor and analyze video.

2) Cryptography (or cryptology; from Greek κρυπτός, "hidden, secret"; and γράφειν, graphein, "writing", or -λογία, -logia, "study", respectively) is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity,

authentication, and non-repudiation. Modern cryptography intersects the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include ATM cards, computer passwords, and electronic commerce.

Cryptography prior to the modern age was effectively synonymous with encryption, the conversion of information from a readable state to apparent nonsense. The originator of an encrypted message shared the decoding technique needed to recover the original information only with intended recipients, thereby precluding unwanted persons to do the same. Since World War I and the advent of the computer, the methods used to carry out cryptology have become increasingly complex and its application more widespread.

Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary. It is theoretically possible to break such a system but it is infeasible to do so by any known practical means. These schemes are therefore termed computationally secure; theoretical advances, e.g., improvements in integer factorization algorithms, and faster computing technology require these solutions to be continually adapted. There exist information-theoretically secure schemes that provably cannot be broken even with unlimited computing power—an example is the one-time pad—but these schemes are more difficult to implement than the best theoretically breakable but computationally secure mechanisms.



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 to the database. Monitoring can be set up to attempt to detect security breaches.

누가 어떤 속성에 접근했으며 무엇이 변하였고 언제 변화가 이루어졌는지에 대한 로깅 레코드는 변한다. 로깅 서비스는 접근 경우와 변화에 대한 레코드를 유지함으로써 데이터베이스의 부검과 같은 일을 할 수 있도록 한다. 때때로 어플-수준의 코드가 데이터베이스에 이러한 흔적을 남기기보다는 변화를 기록하는데 사용된다. 모니터링은 보안 위반을 탐지하기 위하여 설치될 수 있다.

8.3 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 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).

데이터베이스 거래는 충돌로부터 회복된 다음에 어떤 수준의 fault tolerance와 데이터 순수성을 도입하는데 사용될 수 있다. 데이터베이스 거래는 업무의 한 단위이며 전형적으로 데이터베이스(예, 데이터베이스의 객체를 읽고, 쓰고, lock을 수집하는 것 등), 데이터베이스와 다른 시스템에서 지원하고 있는 추상에 대한 수많은 작업을 포함하고 있다. 각각의 거래는 그러한 거래에 (특별한 거래 명령어를 통하여 거래 프로그래머에 의해 결정된) 어떠한 프로그램/코드의 수행이 포함되었는지와 관련해서 그 영역이 잘 정의되어 있다

1) A lock, as a read lock or write lock, is used when multiple users need to access a database concurrently. This prevents data from being corrupted or invalidated when multiple users try to read while others write to the database. Any single user can only modify those database records (that is, items in the database) to which they have applied a lock that gives them exclusive access to the record until the lock is released. Locking not only provides exclusivity to writes but also prevents (or controls) reading of unfinished modifications (AKA uncommitted data).
A read lock can be used to prevent other users from reading a record (or page) which is being updated, so that others will not act upon soon-to-be-outdated information.

The acronym ACID describes some ideal properties of a database transaction: Atomicity, Consistency, Isolation, and Durability.

대문자어 ACID는 데이터베이스 거래의 어떤 이상적인 성질을 설명하고 있다: 원자가, 일관성, 독립성, 인내성.

1) In database systems, atomicity (or atomicness; from Greek a-tomos, undividable) is one of the ACID transaction properties. In an atomic transaction, a series of database operations either all occur, or nothing occurs. A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright. In other words, atomicity means indivisibility and irreducibility.

The etymology of the phrase originates in the Classical Greek concept of a fundamental and indivisible component; see atom.

An example of atomicity is ordering an airline ticket where two actions are required: payment, and a seat reservation. The potential passenger must either:

1. both pay for and reserve a seat; OR
2. neither pay for nor reserve a seat.

The booking system does not consider it acceptable for a customer to pay for a ticket without securing the seat, nor to reserve the seat without payment succeeding.

Another example: If one wants to transfer some amount of money from one account to another, then he/she would start a procedure to do it. However, if a failure occurs, then due to atomicity, the amount will either be transferred completely or will not even start. Thus atomicity protects the user from losing money due to a failed transaction.

2) In database systems, a consistent transaction is one that starts with a database in a consistent state and ends with the database in a consistent state. Consistent state means that there is no violation of any integrity constraints. Consistency may temporarily be violated during execution of the transaction, but must be corrected before changes are permanently committed to the database. If the transaction would leave the database in an illegal state, it is aborted and an error is reported.

Consistency is one of the ACID properties that ensures that any changes to values in an instance are consistent with changes to other values in the same instance. A consistency constraint is a predicate on data which serves as a precondition, post-condition, and transformation condition on any transaction. The database management system (DBMS) assumes that the consistency holds for each transaction in instances. On the other hand, ensuring this property of the transaction is the responsibility of the user.

3) In database systems, isolation is a property that defines how/when the changes made by one operation become visible to other concurrent operations. Isolation is one of the ACID (Atomicity, Consistency, Isolation, Durability) properties.

*Concurrency control

Concurrency control comprises the underlying mechanisms in a DBMS which handles isolation and guarantees related correctness. It is heavily utilized by the database and storage engines (see above) both to guarantee the correct execution of concurrent transactions, and (different mechanisms) the correctness of other DBMS processes. The transaction-related mechanisms typically constrain the database data access operations' timing (transaction schedules) to certain orders characterized as the serializability and recoverability schedule properties. Constraining database access operation execution typically means reduced performance (rates of execution), and thus concurrency control mechanisms are typically designed to provide the best performance possible under the constraints. Often, when possible without harming correctness, the serializability property is compromised for better performance. However, recoverability cannot be compromised, since such typically results in a quick database integrity violation.

Two-phase locking is the most common transaction concurrency control method in DBMSs, used to provide both serializability and recoverability for correctness. In order to access a database object a transaction first needs to acquire a lock for this object. Depending on the access operation type (e.g., reading or writing an object) and on the lock type, acquiring the lock may be blocked and postponed, if another transaction is holding a lock for that object.

*Isolation levels

Of the four ACID properties in a DBMS (Database Management System), the isolation property is the one most often relaxed. When attempting to maintain the highest level of isolation, a DBMS usually acquires locks on data or implements multiversion concurrency control, which may result in a loss of concurrency. This requires adding logic for the application to function correctly.

Most DBMSs offer a number of transaction isolation levels, which control the degree of locking that occurs when selecting data. For many database applications, the majority of database transactions can be constructed to avoid requiring high isolation levels (e.g. SERIALIZABLE level), thus reducing the locking overhead for the system. The programmer must carefully analyze database access code to ensure that any relaxation of isolation does not cause software bugs that are difficult to find. Conversely, if higher isolation levels are used, the possibility of deadlock is increased, which also requires careful analysis and programming techniques to avoid.

4) In database systems, durability is the ACID property which guarantees that transactions that have committed will survive permanently. For example, if a flight booking reports that a seat has successfully been booked, then the seat will remain booked even if the system crashes.

Durability can be achieved by flushing the transaction's log records to non-volatile storage before acknowledging commitment.

In distributed transactions, all participating servers must coordinate before commit can be acknowledged. This is usually done by a two-phase commit protocol.

Many DBMSs implement durability by writing transactions into a transaction log that can be reprocessed to recreate the system state right before any later failure. A transaction is deemed committed only after it is entered in the log.

8.4 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 move, 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 in spite of the fact that tools may exist to help migration between specific DBMS. Typically a DBMS vendor provides tools to help importing databases from other popular DBMSs.

하나의 DBMS와 함께 마련된 데이터베이스는 다른 DBMS로 이동할 수 없다(다시 말해서,

다른 DBMS에서 그것을 기동시킬 수 없다). 그렇지만, 어떤 상황에서 하나의 DBMS로부터 다른 것으로 데이터베이스를 이동시키거나 이주시키길 원할 수 있다. 그 이유는 기본적으로 (서로다른 DBMS는 서로 다른 TCOs를 가질 수 있기 때문에) 경제적, 기능적, 그리고 운영적(서로 다른 DBMSs는 서로 다른 성능을 가질 수 있다)인 것이다. 변형은 만일 가능하다면 그 데이터베이스와 관련된 어플(다시 말해서 관련된 모든 어플 프로그램)을 손대지 않고 이루어져야 한다. 그러므로 그 데이터베이스의 개념적 그리고 외적 구조 차원은 그러한 변형안에서도 유지되어야 한다. 또한 구조적 내적 차원의 몇몇 요소들은 유지되기를 원할 수도 있다. 복잡한 대규모의 데이터베이스 이주는 그 자체로 복잡하고 값비싼 (일회성) 프로젝트일 수 있으며, 이주 결정 시에 이 점이 고려되어야 한다. 이러한 사실에도 불구하고 특별한 DBMS 간의 이주를 지원하는 많은 도구가 존재하고 있으며, 전형적으로 DBMS 상인은 다른 인기 있는 DBMS로부터 데이터베이스를 수입하는 것을 돕는 도구들을 지원하고 있다.

1) Total cost of ownership (TCO) is a financial estimate intended to help buyers and owners determine the direct and indirect costs of a product or system. It is a management accounting concept that can be used in full cost accounting or even ecological economics where it includes social costs.

8.5 Database building, maintaining, and tuning

After designing a database for an application, the next stage is building the database. Typically an appropriate general-purpose DBMS can be selected to be utilized for this purpose. A DBMS provides the needed user interfaces to be utilized by database administrators to define the needed application's data structures within the DBMS's respective data model. Other user interfaces are used to select needed DBMS parameters (like security related, storage allocation parameters, etc.).

어플용 데이터베이스를 디자인한 다음에, 그 다음 단계는 데이터베이스를 구축하는 것이다. 전형적으로 올바른 범용 DBMS는 이런 목적을 실현하기 위하여 선택될 수 있다. DBMS는 필요로 하는 이용자 인터페이스를 제공하여 DBMS의 각각의 데이터 모델에 들어 있는 필요한 어플의 데이터 구조를 정의하도록 데이터베이스 관리자에 의해 활용될 수 있다. 다른 이용자 인터페이스는 필요한 DBMS의 매개변수(보안관련 매개변수, 저장할당량 매개변수 등)를 선택하는데 이용된다.

When the database is ready (all its data structures and other needed components are defined) it is typically populated with initial application's data (database initialization, which is typically a distinct project; in many cases using specialized DBMS interfaces that support bulk insertion) before making it operational. In some cases the database becomes operational while empty of application data, and data is accumulated during its operation.

데이터베이스가 준비되면(모든 그것의 데이터 구조와 기타 필요한 구성요소가 정의되면), 그것을 운영하기 전에 전형적으로 시작용 어플 데이터(전형적으로 하나의 분명한 프로젝트인 데이터베이스 초기화; 많은 경우에 대량의 입력을 지원하는 전문화된 DBMS 인터페이스를 사용한다)를 살도록 하여야 한다. 어떤 경우에 그 데이터베이스는 어플 데이터 없이도 운영이 되며 데이터는 그것이 운영하는 동안에 축적된다.

After the database is created, initialised 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. Databases are often confused with spreadsheets such as Microsoft Excel (Microsoft Access is a database management system, Excel is a spreadsheet program). Both can be used to store information, however a database is more efficient and flexible at storing large amounts of data.

데이터베이스가 만들어지고, 초기화되고, 필요한 데이터가 입력된 다음, 그것은 유지 관리되어야 한다. 여러 가지 데이터베이스 변수가 바뀔 필요가 있을 수 있으며, 그 데이터베이스는 보다 성능을 위하여 튜닝 되어야 한다; 어플의 데이터 구조는 변하거나 추가될 수 있으며 새로운 관련된 어플 프로그램이 그 어플의 기능 등에 추가될 수 있다. 데이터베이스는 종종 Microsoft Excel과 같은 스프레드시트와 혼돈된다. Microsoft Access는 데이터베이스 관리 시스템이며, Excel은 스프레드시트 프로그램이다. 둘 다 정보를 저장하는데 사용할 수 있지만 데이터베이스는 대량의 데이터를 저장하는데 있어서 더욱 더 효율적이며 융통성을 가지고 있다.

1) Database tuning describes a group of activities used to optimize and homogenize the performance of a database. It usually overlaps with query tuning, but refers to design of the database files, selection of the database management system (DBMS) application, and configuration of the database's environment (operating system, CPU, etc.).

Database tuning aims to maximize use of system resources to perform work as efficiently and rapidly as possible. Most systems are designed to manage their use of system resources, but there is still much room to improve their efficiency by customizing their settings and configuration for the database and the DBMS.

Below is a simple comparison of spreadsheets and databases.

다음은 스프레드시트와 데이터베이스를 간단히 비교한 것이다.

Spreadsheet strengths	Spreadsheet Weaknesses
Very simple data storage Relatively easy to use Require less planning	Data integrity problems, including inaccurate, inconsistent and out of date data and formulas. Difficult to validate data e.g. an incorrect formula

Database strengths	Database Weaknesses
Methods for keeping data up to date and consistent	Require more planning and designing
Data is of higher quality than data stored in spreadsheets	Harder to change structure once database is built
Good for storing and organizing information.	Requires more technical knowledge to administrate

8.6 Backup and restore

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 this state is needed, i.e., 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 utilized to restore that state.

때때로 여러 가지 이유로 이전 상태로 되돌려 놓으려는 경우가 발생한다. 예를 들어 데이터베이스가 소프트웨어의 에러로 인하여 문제가 발생했거나 잘못된 데이터로 갱신을 한 경우이다. 이러한 목적으로, 각각의 원하는 데이터베이스 상태가 전용 백업 파일을 통해 유지될 수 있도록(다시 말해서, 데이터베이스의 데이터 구조에 들어있는 데이터와 그것들의 내재적 값) 백업 기능이 경우에 따라 또는 지속적으로 이루어지고 있다. 많은 기술이 이러한 것을 효과적으로 할 수 있도록 존재하고 있다. 이러한 상태가 필요할 때, 다시 말해서 데이터베이스 관리자가 이러한 상태로 데이터 백업을 하기로 결정했을 때(예를 들어, 데이터베이스가 이런 상태에 있을 때 원하는 시점의 상태를 지정함으로써), 이러한 파일들은 그 상태로 회복하도록 활용된다.

8.7 Other

Other DBMS features might include:

Database logs

Graphics component for producing graphs and charts, especially in a data warehouse system

Query optimizer - Performs query optimization on every query to choose for it the most 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.

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.

9. References

1. Jeffrey Ullman 1997: First course in database systems, Prentice-Hall Inc., Simon &Schuster, Page 1, ISBN 0-13-861337-0.
2. Tsitchizris, D. C. and F. H. Lochovsky (1982). Data Models. Englewood-Cliffs, Prentice-Hall.
3. Beynon-Davies P. (2004). Database Systems 3rd Edition. Palgrave, Basingstoke, UK. ISBN 1-4039-1601-2
4. Raul F. Chong, Michael Dang, Dwaine R. Snow, Xiaomei Wang (3 July 2008). "Introduction to DB2". Retrieved 17 March 2013.. This article quotes a development time of 5 years involving 750 people for DB2 release 9 alone
5. C. W. Bachmann (November 1973), "The Programmer as Navigator", CACM (Turing Award Lecture 1973)
6. "database, n". OED Online. Oxford University Press. June 2013. Retrieved July 12, 2013.
7. Codd, E.F. (1970)."A Relational Model of Data for Large Shared Data Banks". In: Communications of the ACM 13 (6): 377–387.
8. William Hershey and Carol Easthope, "A set theoretic data structure and retrieval language", Spring Joint Computer Conference, May 1972 in ACM SIGIR Forum, Volume 7, Issue 4 (December 1972), pp. 45-55, DOI=10.1145/1095495.1095500
9. Ken North, "Sets, Data Models and Data Independence", Dr. Dobb's, 10 March 2010
10. Description of a set-theoretic data structure, D. L. Childs, 1968, Technical Report 3 of the CONCOMP (Research in Conversational Use of Computers)

Project, University of Michigan, Ann Arbor, Michigan, USA

11. Feasibility of a Set-Theoretic Data Structure : A General Structure Based on a Reconstituted Definition of Relation, D. L. Childs, 1968, Technical Report 6 of the CONCOMP (Research in Conversational Use of Computers) Project, University of Michigan, Ann Arbor, Michigan, USA
12. MICRO Information Management System (Version 5.0) Reference Manual, M.A. Kahn, D.L. Rumelhart, and B.L. Bronson, October 1977, Institute of Labor and Industrial Relations (ILIR), University of Michigan and Wayne State University
13. Interview with Wayne Ratliff. The FoxPro History. Retrieved on 2013-07-12.
14. Development of an object-oriented DBMS; Portland, Oregon, United States; Pages: 472 – 482; 1986; ISBN 0-89791-204-7
15. "DB-Engines Ranking". January 2013. Retrieved 22 January 2013.
16. "TeleCommunication Systems Signs up as a Reseller of TimesTen; Mobile Operators and Carriers Gain Real-Time Platform for Location-Based Services". Business Wire. 2002-06-24.
17. Graves, Steve. "COTS Databases For Embedded Systems", Embedded Computing Design magazine, January 2007. Retrieved on August 13, 2008.
18. Argumentation in Artificial Intelligence by Iyad Rahwan, Guillermo R. Simari "OWL DL Semantics". Retrieved 10 December 2010.
19. itl.nist.gov (1993) Integration Definition for Information Modeling (IDEFIX). 21 December 1993.
20. Chapple, Mike. "SQL Fundamentals". Databases. About.com. Retrieved 2009-01-28.
21. "Structured Query Language (SQL)". International Business Machines. October 27, 2006. Retrieved 2007-06-10.
22. Wagner, Michael (2010), "1. Auflage", SQL/XML:2006 - Evaluierung der Standardkonformität ausgewählter Datenbanksysteme, Diplomica Verlag, ISBN 3-8366-9609-6