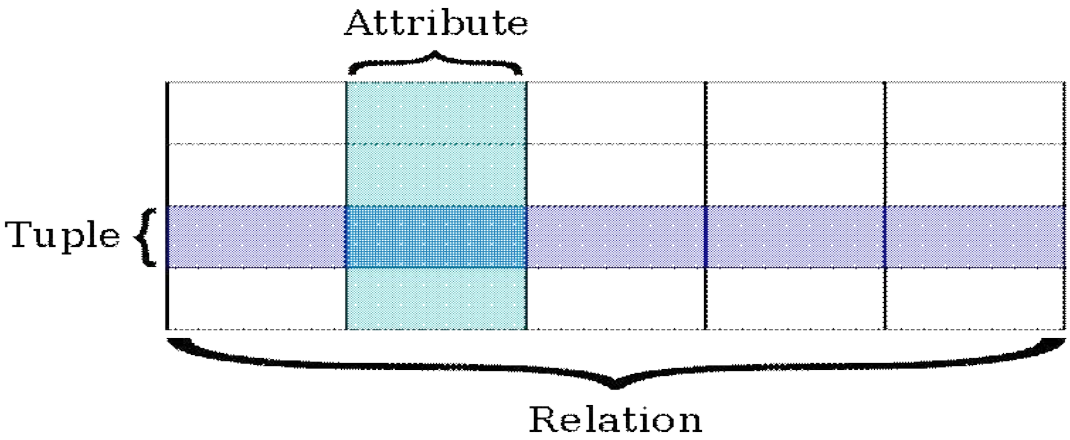
Database

From Wikipedia



2015

대구대 사회과학대학 문헌정보학과

Contents

9.6 Backup and restore

9.7 Other

<References>

<부록> List of Academic Databases and Search Engines

1. Terminology and overview

데이터베이스란 데이터의 조직화 집단이다. 데이터는 정보를 필요로 하는 절차를 지원하는 방식으로 실체를 적절한 모습으로 모델하기 위하여 전형적으로 조직된다. 예를 들어, 빈방찾기를 지원하는 방식으로 호텔 객실의 이용가능성을 모델링하는 것.

DBMS는 데이터를 수집하고 분석할 수 있도록 특히 이용자, 또 다른 어플, 그리고 데이터베이스 그 자체와 상호작용하기 위한 어플들로 디자인된다. 범용 DBMSs는 데이터베이스의 정의, 제작, 질문, 갱신, 그리고 운영하도록 디자인된 소프트웨어 시스템이다. 잘 알려진 DBMSs으로는 MySQL, MariaDB, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, SAP, dBASE, FoxPro, IBM DB2, LibreOffice Base 그리고 FileMaker Pro가 있다. 데이터베이스는 다른 DBAMSs로 이동할 수 없는 것이 일반적이지만, 서로 다른 DBMSs라 하더라도 하나의 어플을 사용하여 복수의 데이터베이스와 작업이 가능하도록 하는 SQL, ODBC, JDBC와 같은 표준을 사용하면 서로 사용할 수 있다.

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.

데이터베이스 이론에는 데이터베이스와 데이터베이스 관리 시스템의 이론적 영역에 대한 연구 및 조사와 관련된 다양한 주제가 포함된다. 그리고 데이터 관리의 이론적 관점에는 여러 분야에 나타나 있는 query languages, computational complexity and expressive power of queries, finite model theory, database design theory, dependency theory, foundations of concurrency control and database recovery, deductive databases, temporal and spatial databases, real time databases, uncertain data and probabilistic databases의 관리, 그리고 Web data가 포함된다.

대부분의 연구가 전통적으로 관계형 모델에 의존하고 있는데, 그 이유는 이 모델이 대부분 어떤 관심사를 가장 간단하고도 가장 기본적으로 모델화할 수 있다고 판단되고 있기 때문이다. 또한 object-oriented 또는 semi-structured models, 최근의 graph data models, 그리고 XML과 같은 또 다른 모델로 얻어지는 결과들은 종종 관계형 모델의 결과로부터 유도되기도 한다.

데이터베이스 이론의 핵심은 쿼리 언어와 이것들의 논리적 관계에 대한 복잡성을 이해하는 것이다. 관계형 대수(algebra)와 Codd's theorem과 대등한 first-order logic 그리고 graph reachability와 같은 중요한 쿼리를 이 언어로는 표현할 수 없다는 것에 대한 깨달음에서 출발하여 datalog와 같이 logic programming과 fixpoint logic을 근거로 더욱 강력한 언어가 연구되었으며, 또 다른 핵심분야는 query optimization 그리고 data integration의 기초에 관한 것이었다. 여기서 대부분의 연구는 chase algorithm을 사용할 때 나타나는 문제 상황에서 조차도 쿼리의 최적화를 가능하게 하는 conjunctive queries에 집중되고 있다.

이 분야에서 이루어지는 주요 연구회의는 the ACM Symposium on Principles of Database Systems (PODS) 그리고 the International Conference on Database Theory (ICDT)이다.

공식적으로 "데이터베이스"란 데이터 그 자체 그리고 데이터 구조를 지원하는 것을 말한다. 다시 말해서, 데이터베이스는 정보의 입력, 저장, 검색, 관리를 위하여 대량의 정보를 운영할 수 있도록 만들어지며, 또한 한 세트(set)로 된 소프트웨어 프로그램들을 사용하여 누구나 모든 데이터에 접근할 수 있도록 제작된다. 그리고 "DBMS"란 이용자가 하나 이상의 데이터베이스와의 접속할 수 있도록 하는 한 벌(suite)로 된 컴퓨터 소프트웨어이다. 이것들이 매우 밀접하게 서로 관련되어 있기 때문에 "데이터베이스"란 용어를 아무 생각없이 사용할 경우에는, 종종 DBMS 그리고 그것이 운영하는 데이터 둘 다를 의미하기도 한다.

전문정보기술 분야를 벗어나면, 데이터베이스란 용어는 때때로 종종 어떤 데이터의 집합을 말하기도 한다(아마도 스프레드쉬트, 또는 심지어 카드색인까지도). 그리고 대부분의 기존 DBMS에서 제공하는 기능은 주로 다음과 같은 4가지이다:

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.

이외에도 DBMS는 저장된 데이터의 순수성과 안전성을 유지하고, 시스템이 잘못될 때 정보를 회복할 책임을 갖는다.

데이터베이스와 그것의 DBMS 둘 다 특정한 데이터베이스 모델의 원칙에 따라야 한다. "데이터베이스 시스템"이란 데이터베이스 모델, 데이터베이스 관리 시스템, 그리고 데이터베이스를 집합적으로 말하기도 한다. 그리고 물리적 측면에서, 데이터베이스 서버란 전용 컴퓨터들을 말하며, 이것들에는 실제로 데이터베이스가 들어 있으며, 단지 그것의 DBMS만이 아니라 관련된 소프트웨어를 기동시키는 역할을 맡고 있다. 데이터베이스 서버는 대부분이 멀티프로세서 컴퓨터들이며, 안정된 저장을 위해 풍부한 메모리와 RAID disk arrays를 갖고 있다. RAID는 어떤 디스크가 깨졌을 때 데이터의 복원을 위해 사용된다. Hardware database accelerators - 고속의 채널을 통해 하나이상의 서버에 연결되어 있음 - 또한 대량의 트랜젝션 처리 환경에서 사용된다. DBMSs는 대부분의 데이터베이스 어플의 중심에 있다. 또한 DBMSs는 내장되어있으면서 네트워크 지원이 가능한 이용자의 다-업무처리용 kernel과 관련해서 설치될 수도 있지만, 현대의 DBMS는 전형적으로 이러한 기능을 제공하기 위하여 표준 운영체제에 의존하고 있다. DBMSs가 중요한 경제적 시장성을 갖게 되면서, 컴퓨터와 저장매체 상인들은 종종 그들 자신의 발전계획에 DBMS의 요구사항을 반영하고 있다.

데이터베이스와 DBMSs는 그것들이 지원하는 데이터베이스 모델(예, 관계형이냐 XML이냐), 기동 가능한 컴퓨터의 종류(server cluster에서부터 휴대전화까지), 데이터베이스 접근용인 쿼리 언어(예, SQL or XQuery), 그리고 performance, scalability(확장성), resilience(복원), and security에 영향을 끼치는 그것들의 구조 공학에 따라 범주화할 수 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.

2. Applications and roles

오늘날 선진국의 대부분 기관들은 자신들의 업무활동을 위해 데이터베이스에 의존하고 있다. 점차적으로, 데이터베이스들은 그 기관의 내부업무를 지원하는데 사용될 뿐만 아니라 ,고객과 납품업자와의 온라인 상호작용에서도 중요하게 사용되고 있다. 데이터베이스는 단지 행정정보를 보관하기 위하여 사용되는 것이 아니라 종종 더욱 전문화된 데이터(예, engineering data or economic models)를 보관하기 위하여 여러 어플들 속에 포함되기도 한다. 데이터베이스 어플의 예로는 computerized library systems, flight reservation systems, computerized parts inventory systems가 있다.

클라이언트-서버 또는 트랜젝션 DBMSs는 많은 이용자가 동시에 그 데이터베이스에 질문하고 갱신할 때 고성능, 이용성, 보안성을 유지하기 위해여 종종 복잡해진다. 그렇지만 개인용이면서 데스크 탑인 데이터베이스 시스템은 복잡성이 다소 떨어지는 경향이 있다. 예를 들어, FileMaker와 Microsoft Access는 내장된 GUI(graphical user interfaces)를 사용하고 있다.

1) **An inventory control system**이란 사물이나 자료의 위치를 파악하고 관리하는 절차이다. 현대 인벤토리 통제 시스템은 종종 barcodes 그리고 radio-frequency identification (RFID) tags를 사용하여 인벤토리 사물의 자 동식별기능을 제공하고 있다.

2) **FileMaker Pro** is a cross-platform relational database application from FileMaker Inc., formerly Claris, a subsidiary of Apple Inc. It integrates a database engine with a GUI-based interface, allowing users to modify the database by dragging new elements into layouts, screens, or forms.

2.1 General-purpose and special-purpose DBMSs

DBMS는 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 개발비가 수많은 이용자에게 분산된다는 사실은 이것들의 선택 시에 비용 대 효과를 가장 우선적으로 고려해야 한다는 것을 의미한다. 그렇지만, 범용 DBMS가 항상 최적의 해결책은 아니다: 어떤 경우에, 범용 DBMS는 불필요한 과부담(overhead)을 불러올 수도 있다. 그러므로 특별한 목적의 데이터베이스를 사용해야하는 시스템에 대한 많은 예가 존재한다. 한 가지 일반적인 예는 이메일시스템이다: 이메일시스템은 이메일 메시지를 최대로 잘 처리하도록 디자인되었으며, 범용 DBMS의 다양한 기능성을 크게 필요로 하진 않는다.

많은 데이터베이스가 최종 이용자를 대신하여 DBMS 인터페이스를 사용하지 않고 그 데이터베이스에 직접 접근할 수 있는 어플 소프트웨어를 가지고 있다. 어플 프로그래머는 직접적으로 wire protocol을 이용하거나 또는 api를 통해 하는 것을 더 많이 할 수도 있다. 데이터베이스 디자이너와 데이터베이스 운영자는 어플 데이터베이스의 구축과 유지를 위하여 전용 인터페이스를 통하여 DBMS와 상호작용하므로, 그들은 DBMS의 운영 방법과 DBMS의 외부 인터페이스 그리고 조절 요소(tuning parameters)에 대하여 더 많은 지식과 이해력을 필요로 한다.

범용 데이터베이스는 대부분이 어떤 기관이나 프로그래머 집단에 의해 개발되었다. 반면에 그것을 사용하기 위한 어플을 다양한 그룹에서 제작하고 있다. 많은 회사에서, 전문 데이터베이스 운영자는 데이터베이스를 관리하고, 보고서를 작성하고, 클라이언트 어플보다는 데이터베이스 그 자체를 기동시키는 코드에 관한 업무를 수행하기도 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).

3. History

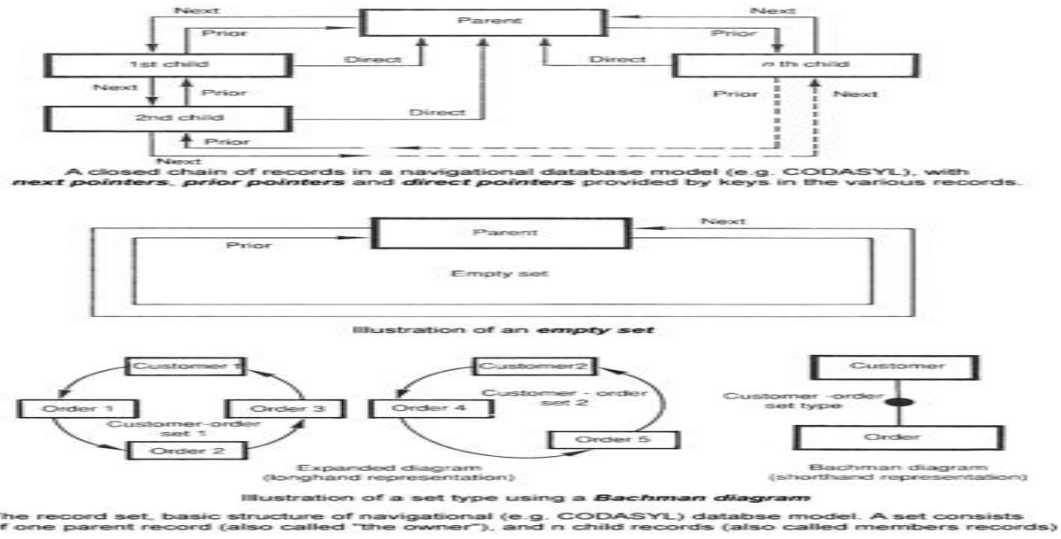
프로세서 분야에서 데이터 처리기술의 진보와 더불어, 컴퓨터 메모리, 컴퓨터 저장 그리고 컴퓨터 네트워크, 데이터베이스의 크기, 용량, 성능 그리고 그것들을 다루는 각각의 DBMSs가 규모면에서 크게 성장하고 있다. 데이터베이스 기술의 발전은 데이터 모델이나 구조에 따라 3 세대로 구분할 수 있다: navigational, SQL/relational, post-relational. 두 가지 주요한 초기 네비게이셔날 데이터 모델은 계층 모델(hierarchical model)들이며, IDMS와 같은 수많은 제품에 설치된 IBM's IMS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

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 for *what* 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.

<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)

Codd에 의해 1970년에 처음으로 제안된 관계형(relational) 모델은 어플이 연결된 링크보다는 콘텐트에 의해 데이터를 탐색해야 한다는 모델이므로, 기존의 전통적 모델과는 차이가 났다. 관계형모델은 원부형태의 테이블(ledger-style tables)로 구성되어 있으며 각각의 테이블은 서로 다른 entity용으로 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 어플에서 사용되었으며, 그것들은 2014년 현재에도 특정분야를 제외하고는 대단한 위세를 떨치고 있다. 이것의 뛰어난 데이터베이스 언어는 관계형 모델용인 표준 SQL이며, 이것은 또한 다른 데이터 모델용의 데이터베이스 언어에도 영향을 끼치고 있다.

사물(Object) 데이터베이스는 1980년대에 object-relational impedance(저항)의 불일치라는 불편함을 개선하기 위하여 개발되었으며, 이것은 "후기 관계형"이라는 용어를 만드는데 일조하였을 뿐만 아니라 혼합형 사물-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 이용가능한 관계형 DBMSs와 비교하여 NoSQL의 고성능을 목표하는 동안, 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. 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

데이터베이스라는 단어의 도입은 1960년대 중반 이전부터 직접접근저장매체(디스크와 드럼)의 사용이 가능하면서 이루어졌다. 그 용어는 과거의 테이프 의존형 시스템과는 대조적으로 사용되었으며, 일상의 배치처리보다는 공유와 쌍방향성의 사용이 가능하다는 것을 의미하였다. 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.

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 approach(방법)로 알려지게 되었고, 곧 수많은 상업적 제품들이 이 방법을 근거로 이용 가능하게 만들어졌다.

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 스캐닝함으로써 발견할 수 있었다. 그 다음의 시스템들은 B-Tree를 추가하여 또 다른 대안적 접근로를 제공하였다. 많은 Codasyl 데이터베이스들 매우 간단한 쿼리 언어를 추가하였지만, 최종 버전에서, CODASYL은 매우 복잡하게 되었으며 유익한 어플을 생산하기 위해서는 더 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.

2) A **hash function** is any function that can be used to map digital data of arbitrary size to digital data of fixed size, with slight differences in input data producing very big differences in output data. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes. One practical use is a data structure called a hash table, widely used in computer software for rapid data lookup. Hash functions accelerate table or database lookup by detecting duplicated records in a large file. An example is finding similar stretches in DNA sequences. They are also useful in cryptography. A cryptographic hash function allows one to easily verify that some input data matches a stored hash value, but makes it hard to reconstruct the data from the hash alone. This principle is used by the PGP algorithm for data validation and by many password checking systems.

IBM 또한 1968년에 자신들만의 DBMS 시스템을 보유하였으며, 이것이 IMS이다. IMS는 System/360에서 사용하기 위하여 아폴로 프로그램용으로 제작된 소프트웨어 였다. IMS는 일반적으로 Codasyl의 개념과 유사했지만, Codasyl의 네트워크 모델 대신에 자체의 데이터 항해용 모델용인 엄격한 계층 모델을 사용하였다. 두 개념 모두 나중에 데이터의 접근 방법에 따라 navigational databases로 알려지게 되었다. 그러나 현재 IMS는 계층적 데이터베이스로 분류되

고 있으며, IDMS와 Cincom Systems' TOTAL 데이터베이스는 네트워크 데이터베이스로 분류되고 있다.

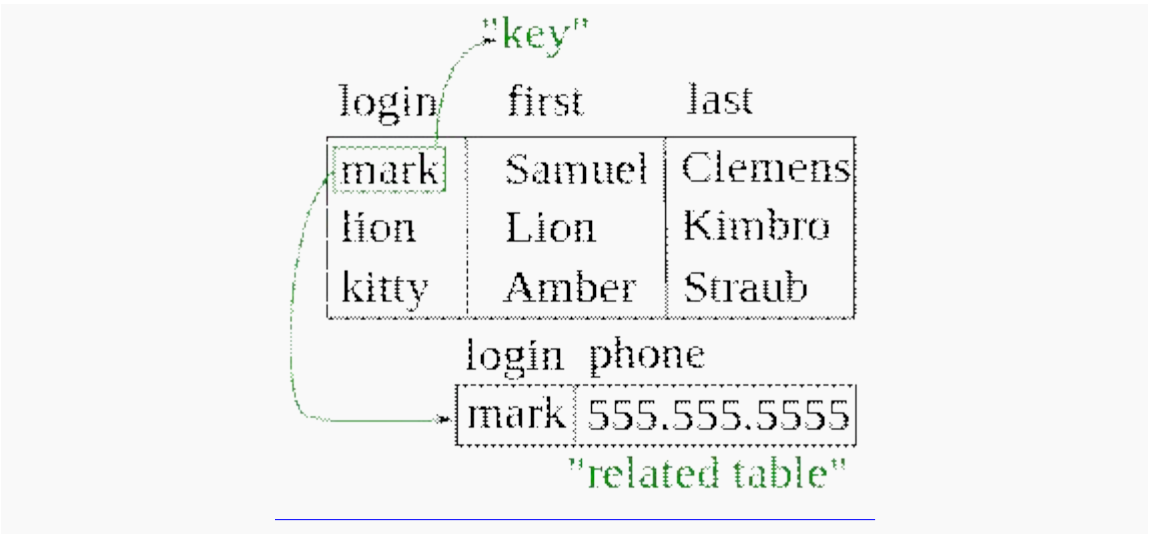
1) The **network model** is a database model conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or lattice.

3.2 1970s relational DBMS

Edgar Codd는 IBM에서 근무할 당시 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 Banks*의 토대가 되었다.

이 논문에서 그는 대용량의 데이터베이스에서 작업하고 저장하는 새로운 시스템을 설명하였다. Codasyl에서처럼 자유로운 형태의 레코드로 되어 있는 몇몇 종류의 링크드 리스트에 레코드를 저장하는 대신에, Codd의 아이디어는 서로 다른 유형의 객체별로 각각의 테이블을 사용하는 고정장 레코드의 테이블을 사용하는 것이었다. 링크드 리스트 시스템은 어떤 한 개의 레코드용의 데이터가 빈공간으로 남아있게 되는 '성근(sparse)' 데이터베이스를 저장할 때 매우 비효율적일 수 있다. 그렇지만 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 이러한 문제를 해결하였다. 결과적으로 데이터는 이러한 테이블 내에서 자유롭게 입력, 삭제, 그리고 편집될 수 있다. 따라서 DBMS는 어플/이용자에세 table view를 제공하는데 필요한 어떠한 유지관리도 가능하게 되었다.

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



관계형 모델은 또한 데이터베이스의 콘텐트로 하여금 링크와 포인터에 대한 지속적인 재작업 없이도 진화하도록 허용하고 있다. 관계형 부분은 전통적인 계층모델처럼 소위 일-대-다로, 그리고 항해 모델처럼 다-대-다로 알려진 다른 객체를 참조하는 객체들로부터 구성된다. 그러므로 관계형 모델은 계층 그리고 항해 모델 둘 다 뿐만 아니라 그것의 고유한 태블러(tabular)

모델도 표현할 수 있는데, 왜냐하면 어플에서 필요로 할 때 이러한 세 가지 모델과 관련된 순수하거나 복합 모델링이 가능하기 때문이다.

예를 들어, 데이터베이스 시스템의 일반적 용도는 이용자, 그들의 이름, 로그인 정보, 다양한 주소와 전화번호에 대한 정보를 추적하는 것이다. 항해 어프로치에서, 모든 이러한 데이터는 하나의 단일 레코드에 포함될 것이지만, 사용되지 않는 아이템은 그 데이터베이스 속에 자리잡지 못할 것이다. 그렇지만 예를 들면, 관계 어프로치에서 데이터는 이용자 테이블, 주소 테이블, 그리고 전화번호 테이블로 정규화될 것이다. 따라서 단지 주소나 전화번호만이 실재로 요구될 때, 이 레코드들이 이들 테이블에서 선택적으로 만들어질 것이다.

뒤로 가서 정보를 함께 링크하는 것이 이 시스템에서 키 이다. 관계형 모델에서, 정보의 어떤 부분(bit)은 "키"로 사용되며, 이것은 특정한 레코드를 유일하게 정의하는데 사용된다. 이용자에 대한 정보가 수집될 때, 선택된 테이블에 저장된 정보를 키로 탐색함으로써 찾을 수 있다. 예를 들어, 이용자의 로그인 이름이 유일하다면, 그 이용자의 주소와 전화번호는 그것의 키로서 로그인 이름과 함께 레코드화 될 것이다. 단일 콜렉션으로 다시 되돌아가 관련된 데이터를 이처럼 간단하게 "재-링킹"하는 것은 전통적인 컴퓨터 언어로서는 결코 디자인할 수 없는 것이다.

항해 어프로치가 레코드를 수집하기 위하여 loop하는 프로그램을 필요로 하는 것과 마찬가지고, 관계 어프로치도 어떤 하나의 레코드에 대한 정보를 수집하기 위하여 loop를 필요로 한다. 필요로 하는 looping에 대한 Codd의 해결책은 집합-지향적(set-oriented) 언어이며, 이것은 나중에 유비쿼터스 SQL을 탄생시키는 제안이었다. tuple calculus로 알려진 수학의 일부를 사용함으로써, 그가 보여준 것은 그 시스템이 단일 운영에 있어서 데이터의 세트를 찾아서 불러오기 위한 단순한 시스템을 제공할 뿐만 아니라 정상적인 데이터베이스의 모든 운영(입력, 갱신 등)을 지원할 수 있다는 것이었다.

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.

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 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.

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.

1970년에, 미시간 대학에서 D.L. Childs' Set-Theoretic Data model을 근거로 MICRO Information Management System의 개발을 시작하였다. Micro는 미국 노동부, 미국 환경보호기구, 그리고 알베르타 대학, 미시간 대학, 그리고 웨인 주립대학의 연구자들에 의해 대량의 데이터 세트를 관리하는데 사용되었다. 이것은 the Michigan Terminal System을 사용하는 IBM의 대형 컴퓨터에서 기동되었다. 이 시스템은 1998년까지 생산되었다.

3.3 Database machines and appliances

1970년대와 1980년대에 통합형 하드웨어와 소프트웨어를 갖춘 데이터베이스 시스템의 구축에 대한 시도가 있었다. 그것의 근저가 되는 철학은 그 같은 통합이 저비용으로 고성능을 제공할 수 있다는 것이었다. 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.

데이터베이스 관리를 위한 하드웨어의 지원과 관련된 또 다른 시도는 ICL의 CAFSaccelerator 인데, 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 때문이었다. 따라서 대부분의 데이터베이스 시스템은 오늘날 범용 컴퓨터 데이터 저장매체를 사용하는 범용 하드웨어에서 기동하는 소프트웨어 시스템들이다. 그렇지만 이러한 아이디어는 아직까지 Netezza와 Oracle(Exadata)과 같은 몇몇 회사에 의해 어떤 어플즈용으로 추구하고 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

IBM은 1970년대 초에 System R처럼 Codd의 개념을 완만하게 적용한 시제품 시스템에서 업무를 시작하였다. 첫 번째 버전은 1974/5년에 나온 다음에, 작업은 (선택 가능한) 레코드용의 모든 데이터가 한 개의 커다란 "chunk(a fragment of information used in many multimedia formats)"에 저장할 필요가 없으므로, 그 데이터를 세분할 수 있는 멀티-테이블 시스템을 시작

하였다. 후속편의 멀티유저 버전은 1978과 1979년에 소비자에 의해 테스트 되었으며, 그 즈음에 표준화된 쿼리 언어인 SQL이 추가되었다. Codd의 아이디어는 Codasyl보다 우수하고 작업도 간단하다는 것이 밝혀졌으며, 그로 인하여 IBM은 SQL/DS로 알려진, 그리고 나중에 Database2(DB 2)로 알려진 System R의 최종 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 Ellison의 Oracle은 System R에 관한 IBM의 논문을 근거로, 서로다른 방향으로부터 출발하였으며, 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 applications 용으로 사용되고 있다. (.org와 .inf domain name registries는 자신들의 중요한 데이터 저장장치로 이것을 사용하고 있으며, 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.

스웨덴에서, Codd의 논문 또한 읽혀졌으며, Uppsala 대학에서 1970년대 중반부터 Mimer SQL을 개발하였으나, 1984년에 이 프로젝트는 개인 기업에 통합되었다. 1980년대 초에, Miner는 어플에서 매우 어려운 일을 처리하기 위한 transaction을 소개하였다. 이 아이디어는 그 후에 대부분의 다른 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 DBMSs, Mimer only implements optimistic concurrency control.

또 다른 데이터 모델인 객체–관계 모델은 1976년에 나타났으며, 초기의 관계형 모델보다 더욱 친숙한 설명에 초점을 맞춤으로써 데이터베이스 디자인용으로 인기를 끌었다. 나중에, 객체–관계 구조는 관계형 모델을 위한 하나의 데이터 모델링 구조로 탈바꿈하였으며, 이 둘 간의 차이는 의미가 없게 되었다.

3.5 1980s desktop databases

1980년대는 desktop computing의 시대를 예고하였다. 새로운 컴퓨터들이 Lotus 1,2,3와 같은 스프레드쉬트와 dBASE와 같은 데이터베이스 소프트웨어를 갖추고 있어서 그것들의 이용자는 더 많은 일을 할 수 있게 되었다. dBASE는 가볍워서 어떤 컴퓨터 이용자도 쉽게 이해할 수 있었다. dBASE 개발자인 C. Wayne Ratliff가 주장: “dBASE는 많은 지겨운 업무에서 전에 사용되었던 BASIC, C, FORTRAN, 그리고 COBOL과 같은 프로그램과는 다르다. 데이터 조작은 이용자에 의해서라기보다는 dBASE에 의해 이루어진다. 그러므로 이용자는 파일을 열고, 읽고, 닫고, 그리고 공간할당을 관리하는 것과 같은 지겨운 일로 혼란을 겪기 보다는 그가 현재 하고 있는 일에 집중할 수 있다.” dBASE는 1980년대와 1990년 초기 최고의 판매 소프트웨어 타이

틀 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

사물 지향형 프로그래밍이 등장하면서 1980년대 내내 다양한 데이터베이스에서 데이터를 처리하는 방법에 대한 발전이 이루어졌다. 프로그래머와 디자이너들은 자신들의 데이터베이스에서 사물처럼 데이터를 취급하기 시작하였다. 만일 어떤 사람의 데이터가 한 데이터베이스에 있다면 그 사람의 속성, 즉 그의 주소, 전화번호, 나이는 외적(extraneous) 데이터로 존재하는 대신에 이제는 그 사람에게 속하는 것으로 여겨지고 있다. 이것은 데이터간의 관계는 사물과 그것들의 속성에 대한 관계이지 개별 필드에 대한 관계는 아니라는 것을 의미한다. object-relational impedance mismatch란 프로그램화된 사물과 데이터베이스 테이블 간에 발생하는 해석(translating)상의 불편함을 말한다. 사물 데이터베이스와 사물-관계 데이터베이스는 프로그래머들이 순수한 관계형 SQL에 대한 대안으로 사용할 수 있는 사물 지향형 언어(때때로 SQL의 확장형태로)를 제공함으로써 이런 문제를 해결하려고 한다. 프로그래밍 측면에서 보면, object-relational mappings (ORMs)로 알려진 라이브러리들(libraries)은 똑같은 문제를 해결하려는 시도이다.

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

2000년대에 후기-관계형 데이터베이스의 차세대는 NoSQL 데이터베이스로 알려졌으며, d 여기에는 신속한 key-value stores와 document-oriented databases가 포함되어 있다. XML 데이터베이스는 한 종류의 정형화된 도큐멘트 지향형 데이터베이스이며, 이것은 XML 도큐멘트 속성들을 근거로 쿼리하는 것을 허용하고 있다. NoSQL 데이터베이스는 종종 매우 빠르며, 고정된 테이블 스키마가 필요하지도 않을 뿐만 아니라 비정규화 데이터를 저장함으로써 join 연산 기능을 피하고, 수평적으로 크기를 디자인하도록 한다.

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.

최근에 고성능의 분할능력(partition tolerance)을 갖춘 대규모의 분산 데이터베이스에 대한 요구가 높아지고 있지만, CAP 공론에 따라, 분산 시스템이 일관성, 이용가능성, 그리고 분할능력을 동시에 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 이용가능성 그리고 분할능력 둘 다에 대한 보증을 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.

가장 인기있는 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 개발된 수많은 새로운 관계형 데이터베이스들을 NewSQL이라 한다.

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

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

5. Database type examples

데이터베이스를 분류하는 한 가지 방법은 그것들의 콘텐트의 종류, 예를 들어 서지, 문서-텍스트, 통계, 또는 멀티미디어 사물과 관련짓는 것이다. 또 다른 방법은 그것들의 응용분야, 예를 들어 회계, 음악, 영화, 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 몇 가지의 형용사를 사용하여 열거하기로 한다.

a) **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 프랫폼은 in-memory 데이터베이스용으로 매우 뜨거운 주제이다. 2012년 5월까지, HANA는 IBM에서 공급한 100TB 주 메모리를 갖춘 서버에서 기동할 수 있

었다. 그 회사의 공동 설립자가 주장하길 그 시스템은 8개의 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.

b) **active** 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". 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.

c) **cloud** 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.

d) data warehouse는 업무 데이터베이스에서 나온 데이터와 종종 시장조사회사와 같은 외부정보원에서 나온 데이터를 보관하고 있으며, 업무데이터에 접근하기 어려운 또 다른 최종 이용자와 관리자가 사용할 수 있는 데이터의 중심 소스가 되었다. 예를 들어, 판매 데이터는 매주 수집하고 합산하기 위하여 UPCs를 사용한 내부 제품코드로 바꿀 수도 있다. 그렇게 함으로써 그것들은 ACNielsen 데이터와 비교할 수 있다. 데이터 보관에 대한 몇 가지 기본적이고 필수적인 구성요소에는 미래의 이용이 가능하도록 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.

e) deductive 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.

3) **Prolog** is a general purpose logic programming language associated with artificial intelligence and computational linguistics. Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations. The language was first conceived by a group around Alain Colmerauer in Marseille, France, in the early 1970s and the first Prolog system was developed in 1972 by Colmerauer with Philippe Roussel.

Prolog was one of the first logic programming languages, and remains the most popular among such languages today, with many free and commercial implementations available. While initially aimed at natural language processing, the language has since then stretched far into other areas like theorem proving, expert systems, games, automated answering systems, ontologies and sophisticated control systems. Modern Prolog environments support creating graphical user interfaces, as well as administrative and networked applications.

f) distributed 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.

g) document-oriented 데이터베이스는 도큐멘트 위주 또는 반-정형화된 데이터의 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.

h) embedded 데이터베이스 시스템은 저장된 데이터에 접근을 요구하는 하나의 어플 시스템으로 조밀하게 통합된 DBMS이다. 그 DBMS는 어플의 최종 이용자에게는 안보이지만, 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.

i) end-user 데이터베이스는 개인별 최종이용자에 의해 개발된 데이터로 구성된다. 이러한 것들의 예로는 documents, spreadsheets, presentations, multimedia, and other files의 콜렉션들이다. 여러 가지 제품이 이러한 데이터베이스를 지원하기 위하여 존 재한다. 이것들 어떤 것은 더 많은 기본적인 DBMS 기능성을 갖춤으로써 full fledges(최종판) DBMSs보다 훨씬 단순하다.

j) federated 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 데 이터베이스로 취급되며, 하나의 통합된 개념적 view를 제공한다.

k) 때때로 multi-database라는 용어는 비록 그것이 하나의 단일 어플에 협력하는 다소 통합력이 떨어진(예를 들어, FDBMS와 어떤 관리 통합 스키마가 없는) 집단의 데이터베이스를 언급하더라도 federated 데이터베이스와 동의어로 사용된다. 이런 경우에 전형적으로 미들웨어는 분배용으로 사용되며, 이것에 전형적으로 관여하고 있는 데이터베이스들 간에는 distributed (global) transaction이 가능하도록, 예를 들어 the two-phase commit protocol과 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.

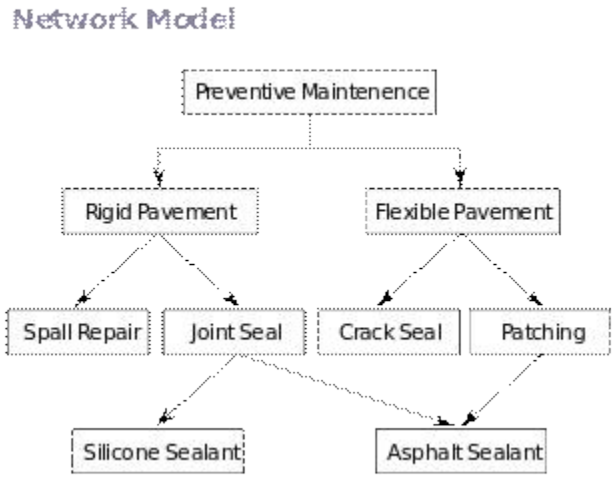
3) A **distributed transaction** is an operations bundle, in which two or more network hosts are involved. Usually, hosts provide transactional resources, while the transaction manager is responsible for creating and managing a global transaction that encompasses all operations against such resources. 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).

l) graph 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 databases 와 같은 전문 그래픽 데이터베이스와는 다르다.

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.



m) hypertext 또는 hypermedia 데이터베이스에서, 예를 들어 다른 텍스트, 그림, 필름과 같은 사물을 표현하는 어떤 단어나 텍스트는 해당 사물로 하이퍼링크될 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. The World Wide Web은 대규모의 분산 하이퍼텍스트 데이터베이스이다.

n) knowledge base(KB, kb)란 지식의 전산화된 수집, 조직, 검색을 위한 수단을 제공하는 지식관리를 위한 특별한 종류의 데이터베이스이며, 또한 문제와 더불어 그것들의 해결책과 관련 경험을 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.

o) 모바일 데이터베이스는 모바일 컴퓨팅 기기에서 운영되거나 동기화될 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.

p) operational 데이터베이스는 조직의 운영에 대한 상세한 데이터를 저장한다. 이것들은 전형적으로 트랜젝션을 이용하여 비교적 많은 양의 정보를 갱신한다. 이것의 예로는 기업고객에 대한 접촉, 신용, 그리고 인구통계적 정보를 기록하고 있는 고객용 데이터베이스, 그리고 제품의 구성요소와 부분별 인벤토리에 대한 상세 정보를 기록하고 있는 기업 자원 기획 시스템에 대한 정보 및 종업원에 대한 월급, 이윤, 기술과 같은 정보를 갖고 있는 인사 데이터베이스, 그리고 끝으로 조직의 자금, 회계, 금융거래를 기록해 놓은 재정 데이터베이스가 있다.

q) parallel 데이터베이스는 데이터를 로딩하고 색인을 만들고 쿼리를 평가하는 것과 같은 업무를 위하여 병렬처리방식(parallelization)으로 성능 개선을 추구한다.

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").

기본적인 하드웨어의 구조로부터 만들어지는 주요한 parallel DBMS 구조는 다음과 같다:

a) Shared memory architecture: 복수의 프로세서가 주 메모리 공간뿐만 아니라 기타 데이터 저장 공간도 공유한다.
b) Shared disk architecture: 디스크를 공유하는 구조: 전형적으로 복수의 프로세서들로 이루어진 각각의 프로세싱 유니트가 자신만의 주 메모리를 갖고 있지만, 모든 유니트들이 다른 저장공간을 공유한다.
c) Shared nothing architecture: 아무것도 공유하지 않는 구조: 각각의 프로세싱 유니트가 자신만의 주 메모리와 다른 저장공간을 갖고 있다.
d) Probabilistic databases: 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.

e) Real-time databases : 곧바로 실행되거나 회수되어야 하는 결과를 충분하고도 신속하게 교신을 통해 실시간으로 처리한다.
f) A spatial database: 다차원적인 모습으로 데이터를 저장할 수 있다. 그런 데이터에 대한 쿼리에는 “이 지역에서 가장 가까운 호텔은?”과 같은 위치기반 쿼리가 포함된다.
g) A temporal database: 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.
h) A terminology-oriented database: 객체지향형 데이터베이스에서 만들어지며, 종종 특정한 분야로 제한된다.
i) An unstructured data database: 관리 및 보호가 가능한 방법으로 일반 데이터베이스에서는 자연스럽게 그리고 편리하게 다룰 수 없는 다양한 사물들을 저장하려는 의도를 가지고 있다. 여기에는 email messages, documents, journals, multimedia objects 등이 포함될 수 있다. 이 이름이 오해를 불러일으킬 수 있는데 왜냐하면 어떤 사물들은 매우 정형화되어있기 때문이다. 그렇지만, 모든 잠재적 사물 집단은 미리 설정된 정형화된 틀에 맞지는 않는다. 대부분의 기존의 DBMSs는 현재 다양한 방법으로 비정형화된 데이터를 지원하고 있으며, 새로운 전용 DBMSs도 나타나고 있다.

6. Database design and modeling

데이터베이스 디자이너의 첫 번째 임무는 데이터베이스에 저장된 정보의 구조를 다룰 개념적 데이터 모델을 만드는 것이다. 이것을 하는 일반적인 방법은 drawing tools를 사용하여 객체-관계 모델(entity-relationship model)을 개발하는 것이다. 또 한가지 인기있는 방법은 the Unified Modeling Language 이다. 성공적인 데이터 모델을 위해서는 모델화하려는 외부 세계의 잠재적 상태를 정확하게 반영하여야 한다: 예를 들어, 만일 사람들이 한 개 이상의 전화번호를 가질 수 있다면, 이런 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 프랑크프르트를 거쳐 두바이까지 간다면, 그것은 한 번의 비행인가 또는 2번이나 3번의 비행인가?"와 같은 질문 – 이 포함된다. 이러한 질문들에 대한 해답들은 객체들(customers, products, flights, flight segments)과 그것들의 관계와 속성용으로 사용된 전문용어의 정의로 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.

때때로 개념적 데이터 모델을 생산하는 데에는 사업 프로세서, 또는 기관의 업무흐름도의 분석에서 발생한 입력요소들이 포함되기도 한다. 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 도움이 될 수 있다.

이용자가 이해하기 쉬운 개념적 데이터 모델을 생산한 다음에, 그 다음 단계는 이것을 데이터베이스에서 적절한 데이터 구조로 실행하는 스키마로 변환시키는 것이다. 이 과정을 종종 논리적 데이터베이스 디자인이라고 부르며, 그 결과는 스키마(schema)의 형태로 표현된 논리적 데이터 모델이다. 개념적 데이터 모델이 적어도 이론적으로는 특정한 데이터베이스 테크놀로지와 상관없다 하더라도, 논리적 데이터 모델은 특정한 DBMS에 의해 지원을 받는 특별한 데이터베이스 모델과 관련해서 표현되어야 한다(data model과 database model이란 용어들은 종종 호환적으로 사용되지만, 이 글에서 특별한 데이터베이스의 디자인을 위해서는 data model을 그리고 그 같은 디자인을 표현하기 위하여 사용되는 모델링 표기(notation)을 위해서는 database model을 사용한다).

범용 데이터베이스용으로 가장 인기있는 데이터베이스 모델은 관계형 모델이며, 더 정확하

게 말해서 SQL 언어로 표현된 관계형 모델이다. 이 모델을 사용하여 논리적 데이터베이스 디자인을 제작하는 과정에서는 정규화(normalization)로 알려진 방법론적 접근방식을 사용한다. 정규화의 목표는 각각의 기본적인 "사실"이 단지 한 곳에만 기록되도록 하여 그것의 추가, 갱신, 그리고 삭제가 자동적으로 일관성 있게 이루어지도록 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.

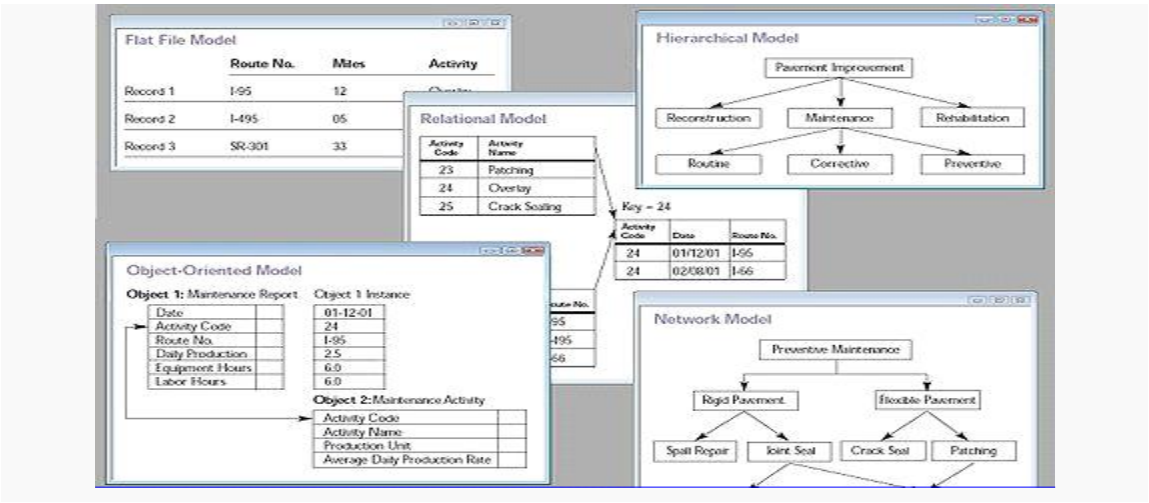
A standard piece of database design guidance is that the designer should first create a fully normalized design; then selective denormalization can be performed for performance reasons.[

데이터베이스 디자인의 최종 단계는 성능, 확장성(scalability), 회복력, 안전성 등에 영향을 끼치는 사안들에 대하여 결정하는 것이다. 이것을 종종 물리적 데이터베이스 디자인이라고 부른다. 이 단계의 중요한 목표는 데이터 독립성(data independence)이다. 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.

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

6.1. Database models



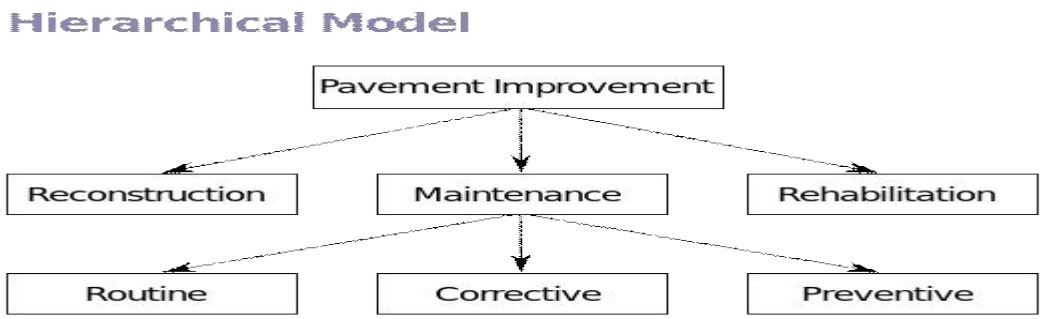
Collage of five types of database models.

데이터베이스 모델이란 기본적으로 어떠한 방식으로 데이터를 저장, 조직, 취급할 것인지를 다루는 데이터베이스의 논리적 구조를 결정하며 데이터 모델의 한 유형이다. 가장 인기 있는 데이터베이스 모델은 테이블을 근거로 하는 형식의 관계형 모델(또는 관계형과 유사한 SQL 모델) 이다.

데이터베이스용의 일반적인 논리적 데이터 모델에는 다음과 같다:

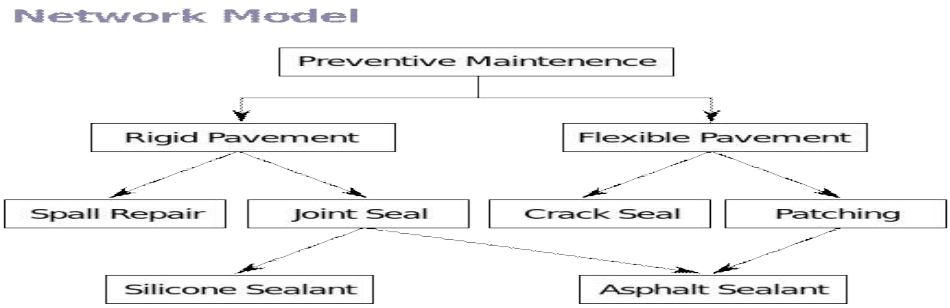
(1) Hierarchical database model

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



(2) Network model

네트워크 모델은 사물과 그것들의 관계를 표현하는데 있어서 유연한 방법으로 인식되어진 데이터베이스 모델이다. 이것의 뛰어난 특징은 그 스키마(사물 유형은 nodes로, 관계 유형은 아크로 표현된 그래프와 같은)가 계층구조라든지 또는 격자(lattice)구조라든지 상관하지 않는다.



(3) Relational model

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

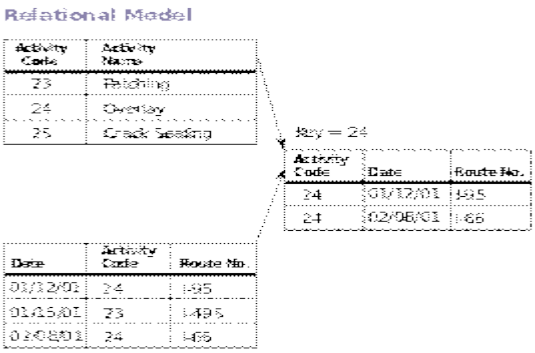
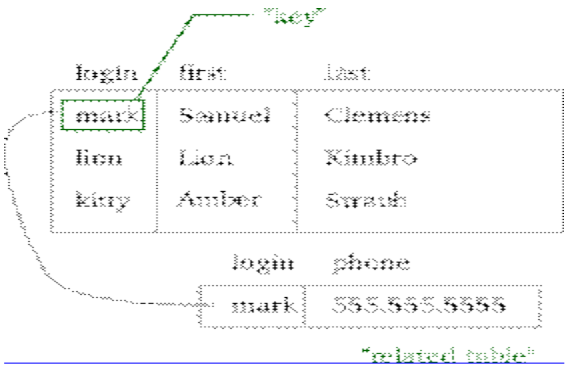


Diagram of an example database according to the Relational model.



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

관계형 모델의 목적은 데이터와 쿼리(queries)를 특정화하기 위한 선언적 방법을 제공하는 것(이용자가 직접 그 데이터베이스에 들어있는 정보가 무엇이고, 그가 그것에서 원하는 정보가 무엇인지를 지정하는 방법), 그리고 그 데이터베이스 관리 시스템 소프트웨어로 하여금 쿼리에 답하도록 하는 검색절차와 데이터를 저장하는 데이터 구조의 기술(describing)을 담당하도록

하는 것이다.

대부분의 관계형 데이터베이스는 SQL 데이터 정의와 쿼리 언어를 사용한다: 즉, 이 시스템들은 소위 관계형 모델과 공학적으로 유사한 것(engineering approximation)을 사용한다. SQL 데이터베이스 스키마의 테이블(table)은 술어적 변수(a predicate variable)에 해당한다; 다시 말해서, 테이블의 콘텐트는 관계에 해당하고; 키 제한조건, 기타 제한조건, SQL 쿼리는 술어(속성)에 해당한다. 그렇지만 SQL 데이터베이스(DB2를 포함하여)는 많은 부분이 관계형 모델의 영향을 받고 있으나, Codd는 본래의 원칙과 타협하는 변종들에 대하여 강력하게 비난하였다.

(4) Entity-relationship model

ER 모델은 데이터베이스를 설명하는 하나의 추상적 방법이다. 테이블에 데이터가 저장되는 관계형 데이터베이스의 경우에, 이들 테이블에 있는 데이터는 다른 데이블에 있는 데이터를 시사한다(point). - 예를 들어, 데이터베이스에 있는 여러분의 항목(entry)은 여러분 자신의 각각의 전화번호를 나타내는 여러 가지 항목을 나타낼 수 있다. ER 모델에서 여러분은 하나의 객체이고, 각 전화번호도 하나의 객체이며, 여러분과 전화번호와의 관계는 'has a phone number'라고 알려줄 수 있다. 이러한 객체들과 관계들을 디자인하도록 만들어진 다이어그램을 객체-관계 다이어그램 또는 ER diagram이라 부른다.

(4-1) Conceptual data model

개념적 데이터 모델은 가장 높은 수준의 ER 모델이며, 그 속에는 최소한의 granular(과립상의) detail을 포함하고 있지만, 무엇이 그 모델 세트에 포함되어야 하는지에 대한 범위를 설정하고 있다. 개념적 ER 모델에서는 보통 대상기관에서 공통으로 사용하는 master reference data 객체를 정의하고 있다. 기업용 개념적 ER 모델을 개발하는 것은 그 기관의 데이터 구조를 기록하는 것을 지원하는데 매우 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.

개념적 ER 모델은 하나 이상의 논리적 데이터 모델의 기초로 사용될 수 있다., 개념적 ER 모델의 목적은 논리적 ER 모델의 집단 사이에서 master data entities를 위한 구조적 메타데이터 공통성(commonality)을 수립하는 것이다. 이러한 개념적 데이터 모델은 데이터 모델의 통합을 위한 기본으로 ER 모델 간의 commonality 관계를 형성하는데도 사용되기도 한다.

(4-2) Logical data model

만일에 논리적 모델의 범위가 단지 분명한 정보시스템의 개발만을 위한 것이라면, 논리적 ER 모델은 개념적 ER 모델을 필요로 하지 않는다. 논리적 ER 모델은 개념적 ER 모델보다 더 많은 details를 포함한다. master data entities에 따라서, 운영과 거래 데이터 객체가 정의된 다음, 각 데이터 객체의 details가 개발되며, 그런 다음에 이 데이터 객체간의 객체 관계가 수립된다. 논리적 ER 모델은 그렇지만 그것을 실행시킬 수 있는 기술과는 독립적으로 개발되어야 한다.

(4-3) Physical data model

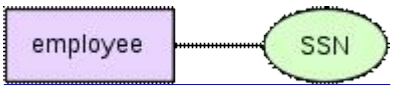
한 개 이상의 물리적 ER 모델이 각각의 논리적 ER 모델로부터 개발될 수 있다. 물리적 ER 모델은 보통 하나의 데이터베이스와 같은 사례로 개발되고 있다. 그러므로 각각의 물리적 ER 모델은 데이터베이스를 생산할 수 있는 충분한 details를 포함하고 있어야 하며, 각각의 물리적 ER 모델은 각각의 데이터베이스 관리 시스템이 다소 차이가 있더라고 기술적으로 독립성을 가져야 한다.

물리적 모델에서는 일반적으로 관계형 데이터베이스 사물(데이터베이스 테이블, unique key indexes와 같은 데이터베이스 색인, 그리고 foreign key constraints나 commonality constraints와 같은 데이터베이스 제한조건)처럼 데이터베이스 관리 시스템 속에 정형화된 메타데이타를 실재로 만들기 위하여 기술보다 앞서가고 있다.

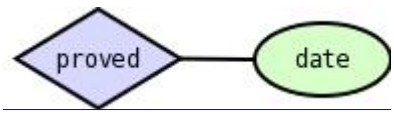
정보시스템 디자인의 첫 번째 단계에서는 데이터베이스에 저장되어야 하는 정보의 유형과 정보의 요구를 기술하기 위하여 필요사항을 분석하는 동안에 이러한 모델들이 사용된다. 데이터 모델링 기법은 특정 관심 분야용의 ontology(다시 말해서, 사용된 용어와 그것들의 관계에 대한 전반적 견해와 분류)를 기술하는데 이용될 수도 있다. 그리고 데이터베이스를 근거로 하는 정보시스템을 디자인하는 경우에, 개념적 데이터 모델은 나중에(대체로 논리적 디자인이라고 부른다) 관계형 모델처럼 논리적 데이터 모델 속에 포함 된다(mapped); 그런 다음에 이것은 물리적 디자인이 이루어지 동안에 물리적 모델에 포함된다. 한 가지 주목할 것은 때때로 이러한 양 쪽 단계 모두를 '물리적 디자인'이라고 부르기도 하며, 또한 이것은 데이터베이스 관리 시스템에서도 사용된다.

** Entity-relationship modelling: Supplement **

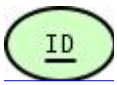
Two related entities



An entity with an attribute



A relationship with an attribute



Primary key

객체란 유일하게 식별가능하며 독립적인 존재로 인식되는 물체(thing)라고 정의할 수 있으며, 복잡한 도메인에서 나온 추상적 개념이다. 우리가 어떤 객체에 대해 말할 때, 우리는 보통 다른 모습과 구별할 수 있는 실재의 모습에 대해 말하는 것이다.

객체는 집이나 차 또는 이벤트(고객의 거래사항과 주문)와 같은 물리적 개념일 수도 있다. 비록 객체라는 용어가 가장 널리 일반적으로 사용되고 있다 하더라도, Chen은 실재로 객체와 객체유형(an entity and an entity-type)을 구분하여야 한다고 주장하고 있다. 하나의 객체-유형은 하나의 범주(category)이다. 엄격하게 말해서, 객체는 특정한 객체-유형의 한 사례(instance)이므로, 대체로 객체-유형에는 많은 사례가 존재한다. 객체-유형이라는 단어가 다소 모호하기 때문에, 대부분의 사람들은 이 용어의 이음동의어(synonym)로 객체를 사용하기도 한다.

객체는 명사로 여겨질 수도 있다. 예를 들어, 컴퓨터, 종업원, 노래, 수학적 이론 등이다. 그리고 관계는 객체가 어떻게 서로 연결되어 있는가를 나타낸다. 관계는 두 개 이상의 명사를 링크시키는 동사로 여겨질 수도 있다. 예를 들어, 회사와 컴퓨터 간의 소유(owns) 관계, 종업원과 부서 간의 감독(supervises) 관계, 예술가와 노래 간의 연주(performs) 관계, 수학자와 정리간의 입증(proved) 관계 등이 있다.

위에서 설명한 모델은 언어학적 측면에서 보면, 선언적 데이터베이스 쿼리 언어인 ERRO로 표현되고 있으며, EERO는 자연어의 구조를 모방하고 있다. ERROL의 어의와 적용(implementation)은 새로운 관계형 대수(RRA; 객체-관계 모델에 적용되고 있으면서 그것의 언어적 요소를 사용하고 있는 관계형 대수)에 의존하고 있다.

객체와 관계는 둘 다 속성을 가질 수 있다. 예를 들어 employee 객체는 Social Security Number(SSN) 속성을 가질 수 있다: 그것의 proved 관계는 date 속성을 가질 수도 있다. 모든 객체는(약한 객체가 아니라면) 유일하게 식별 가능한 속성을 최소한으로 갖고 있어야 하며, 이것을 그 객체의 으뜸키(primary key)라고 부른다. 객체-관계 다이어그램은 단일 객체나 단일 relations의 사례를 보여주지 않는다. 그 보다는 이것들은 객체 세트와 관계 세트를 보여준다. 예를 들어, 특별한 *song*은 객체이다. 데이터베이스에서 모든 songs의 집단은 하나의 객체 세트이다. 어린이와 그의 점심 간의 존재하는 *eaten* 관계는 단일 관계이다. 데이터베이스에서 이러한 모든 어린이-점심 관계의 세트는 하나의 관계 세트이다. 다시 말해서, 하나의 관계 세트는 수학에서 하나의 relation을 말하지만, 하나의 관계는 그 relation의 한 멤버를 말한다. 그리고 관계 세트에는 cardinality constraints가 존재할 수 있다.

a) Mapping natural language

Chen은 다음과 같은 ER 다이어그램에서 자연어 설명을 나타내는 'rules of thumb'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

b) Relationships, roles and cardinalities

Chen의 자신의 글에서, 관계와 그것의 역할에 대한 예를 제시하였다. 그는 “marriage” 관계와 그것의 두 가지 역할로 “husband”와 “wife”를 설명하고 있다. 한 사람은 결혼(관계)에서 남편의 역할을 하고, 다른 사람은 (동일한) 결혼에서 부인 역할을 한다. 이들 단어들은 명사이다. 놀랄 것도 없이 사물을 명명(namining)하는 데는 명사가 필요하다.

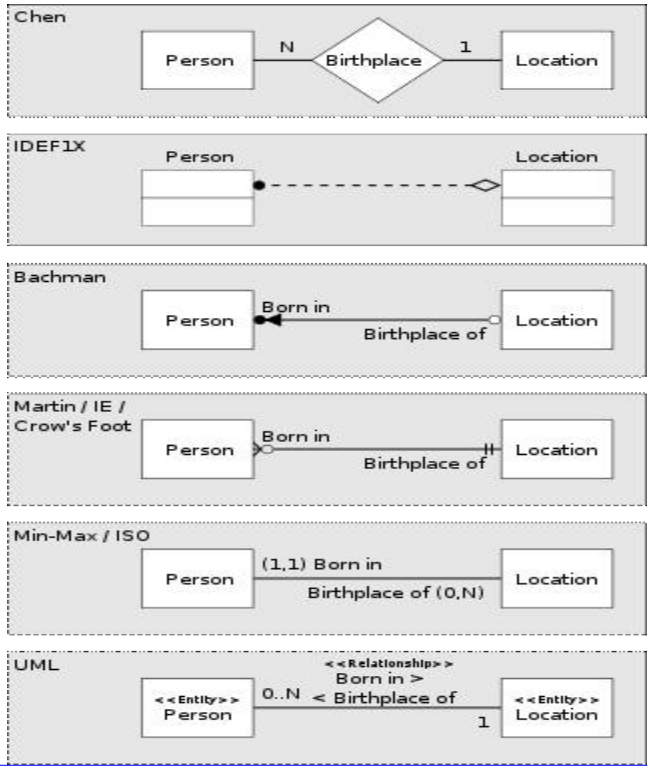
아무리 새로운 아이디어가 일상적이라 하더라도, 많은 노력이 새로운 용어를 위해 이루어졌으며, 이러한 용어가 오래된 아이디어에 적용되었다. 그러므로 이러한 다이어그램의 lines, arrows, crows-feet는 Chen의 관계형 다이어그램보다도 그 이전의 Bachman diagram에 더 많은 영향을 받고 있으며, 특히 동사와 어구로서 관계와 역할을 “name”하는 것이 유행 되었다 (지금은 거의가 독점적으로 사용되고 있다).

c) Role naming

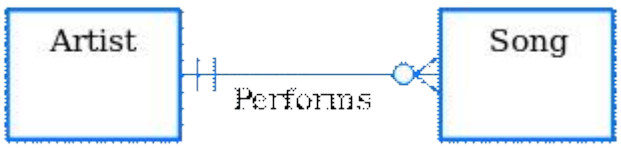
어구를 사용하여 역할을 명명하는 것이 보편적이다. 예를 들면, is-the-owner-of 와 is-owned-by 등이며, 이럴 경우에 정확한(correct) 명사는 “owner”와 “possession”이다. 그러므로 "person plays the role of is-the-owner-of" etc. 보다는 "person plays the role of owner" 와 "car plays the role of possession" 이다.

명사의 사용은 어의적 모델로부터 물리적 implementations을 이루고자할 때 직접적으로 도움이 된다. 어떤 사람이 차와 함께 두 개의 관계를 가질 때, 즉시 그 의미를 알 수 있도록 “owner_person”과 “driver_person”처럼 매우 간단한 이름으로 나타낼 수 있다.

d) Cardinalities



1) 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.



1) 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).

객체-관계 모델링의 Chen 표기법은 객체세트를 표현하기 위하여 rectangles를, 그리고 자체적으로 속성과 관계를 가질 수 있는 first-class objects용으로 적합한 관계를 표현하기 위하여 diamonds를 사용한다. 만일 객체 세트가 관계 세트에 연결된다면, 그것들은 선으로 표현된다. 속성은 ovals로 그려지며, 한 객체나 관계 세트에 직접 선으로 연결된다.

카디낼러티 제한조건은 다음과 같다:

1) 이중선은 참여조건을 나타낸다; 즉, totality 또는 surjectivity(전사성). 그리고 객체 세트에 있는 모든 객체는 관계 세트에 있는 적어도 하나의 관계에 참가하여야 한다.

2) 객체세트로부터 관계세트로 표시된 화살표는 키 제한조건, 즉 injectivity(최소참여)를 나타

낸다. 객체세트의 각 객체는 많아야 관계 세트에 있는 하나의 관계에 참가할 수 있다.

3) 굵은 선은 양쪽, 다시 말해서 bijectivity(완전참여)를 나타낸다. 객체 세트에 있는 각 객체는 정확하게 하나의 관계 속에 포함되어야 한다.

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

속성은 종종 다이어그램을 어지럽히기 때문에 생략기도 한다: 어떤 다이어그램 기법에서는 종종 객체 세트용으로 그린 사각형 속에 객체의 속성을 열거하기도 한다.

<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

e) Crow's Foot Notation

Crow's Foot notation은 Barker's Notation, SSADM and Information Engineering에서 사용되었다. Crow's Foot diagrams 은 boxes로 entities를, boxes간의 라인으로 관계를 표현하고 있다. 이러한 라인의 양쪽 끝에 서로 다른 모양이 있는데 이것은 이 관계의 카디넬리티를 나타낸다.

f) ER diagramming tools

많은 ER 다이어그램 도구가 존재한다. ER 모델과 SQL을 해석하여 생산할 수 있는 그리고 데이터베이스를 분석할 수 있는 무료 소프트웨어 ER 다이어그래밍 도구들은 MySQL Workbench(전에는 DBDesigner)와 Open ModelSphere(open-source) 이다. 데이터베이스와 application layer code (webservices)를 생산할 수 있는 ER 도구는 the RISE Editor이다. 또한 비록 전문적이지만 SQL Power Architect 또한 무료 버전을 갖고 있다.

전문 ER diagramming tools로는 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 등이 있다.

Free software diagram tools는 단지 그것에 대한 어떠한 지식없이도 모양(shapes)을 그

릴 수 있으나 SQL을 생산하지는 못한다. 이러한 것들의 예로는 Creately, yEd, LucidChart, Calligra Flow, and Dia 등이 있다.

g) ER and semantic modelling

ER 모델링의 아버지인 Peter Chen은 자신의 세미나 논문에서 말했다: “객체-관계 모델에는 실세계가 객체와 관계로 구성된다는 보다 자연스러운 견해를 반영하고 있으므로, 이것은 실세계에 대한 어떤 중요한 어의적 정보를 갖고 있다.” 그는 여기서 고대 그리스 철학자인 Socrates, Plato and Aristotle (428 BC) 시대에서부터 현대의 Peirce, Frege and Russell의 epistemology(인식론), semiotics(기호학) and logic까지의 철학적 및 이론적 전통에 따랐다. Plato는 스스로 지식을 변하지 않는 Forms(Socrates에 따르면, forms란 대체로 많은 유형의 실체와 성질에 대한 추상적 표현이나 archetypes(원형)을 말한다.)와 그것들 간의 관계의 이해하는 것과 연관시키고 있다. 자신의 1976년 기사에서, Chen은 분명하게 entity-relationship diagrams 과 record modelling techniques의 차이를 말하고 있다: “The data structure diagram은 레코드를 조직적으로 표현한 것이지, 객체와 관계를 정확하게 표현한 것은 아니다.” 여러 다른 학자들이 그의 프로그램을 지지하고 있다.

어의적 모델은 개념의 모델이며, 때때로 "platform independent model"이라고도 부른다. 이것은 내포적(intensional) 모델이며, Carnap이래로 가장 최근에 다음과 같은 것으로 잘 알려지게 되었다: “개념의 완전한 의미는 두 가지 관점으로 구성 된다: 그것의 intension(내포)와 extension(외연). 첫 번째 것은 전체적인 개념의 집단에 박혀있는 하나의 개념을 의미한다: 다시 말해서, 다른 개념에 대한 모든 relations의 totality이다. 두 번째는 개념에 대한 참조적인(referential) 의미를 수립한다.: 다시 말해서, 실재의 또는 잠재적 세상에 있는 그것의 상대방(counterpart)이다.”

외연적 모델은 특수한 방법이나 기술의 요소에 대하여 작성되므로 "platform specific model" 이다. 분명하게 UML specification에서 언급하길, class models에서의 associations는 외연적인 것으로, 이것은 사실상 그 스펙에 의해 제공된 외향적으로 추가된 그리고 그리고 이전의 후보인 몇몇 “semantic modelling languages"에서 제공한 “adornment(장식)”을 고려해 보면 명확하게 알 수 있다.

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.

h) Limitations

1) ER 모델은 관계형 데이터베이스에서 쉽게 정보의 콘텐트를 표현할 수 있다고 가정된고 있다. 이것들은 단지 이러한 정보의 관계적 구조만을 표현한다.

2) 이것들은 반쯤-정형화된(semi-structured) 데이터와 같이 쉽게 관계형 형태로 표현할 수 없는 정보가 들어 있는 시스템에서는 부적당하다.
3) 많은 시스템에서, 내장된 정보의 변화 가능성은 명확하게 스펙에서 충분히 보장하여야 한다.
4) 어떤 저자들은 구조적으로 변화를 나타낼 수 있도록 ER 모델링을 확장시켰으며, 이러한 시도는 원저자로부터 지지를 받았다: 그 예가 Anchor Modelling이다. a process modeling technique을 사용하여 독립적으로 변화를 모델하는 것도 하나의 대안이다. 또 다른 기법들이 시스템의 또 다른 요소용으로 추가적으로 사용될 수 있다. 예를 들어, ER 모델은 UML에서 제공하는 14가지의 다양한 모델링 기법 중 단지 하나에 불과하기 때문이다.
5) ER 모델링은 scratch로부터 정보를 특정화시키는 것이 목표이다. 이것은 새롭고, 독립적인 정보시스템을 디자인하는데 적격이지만, 이미 자신의 데이터 표현을 상세하게 정의하고 있는 기존의 정보자원을 통합하는데 있어서는 별로 도움이 되지 않는다.
6) 비록 원칙적으로 적합하더라도, ER 모델링은 거의가 하나의 독립된 행위로 사용되지는 않는데, 한 가지 이유는 오늘날 관계형 데이터베이스 관리 시스템에 직접적으로 프로그램과 디자인을 지원하는 도구가 풍부해 졌기 때문이다. 이들 도구들에서 기존의 데이터베이스의 ER 다이어그램과 매우 비슷한 데이터베이스 다이어그램을 쉽게 발췌할 수 있으며, 이것들은 그 같은 다이어그램에 있는 정보에 대한 대안적 뷰(views)를 제공한다.
7) 서베이를 통해, Brodie와 Liu는 100개의 회사 샘플로부터 객체-관계 모델의 사례를 찾을 수 없었다. Badia와 Lemire는 안내정보의 부족 때문에 뿐만 아니라 데이터 통합의 지원 부족과 같은 이익의 부족으로 사용의 부족이 나타났다고 비난했다.
8) is-a 관계와 같은 사물-지향형과 밀접하게 관련된 enhanced entity-relationship model (고급형 ER)에서는 ER 모델링에서 표현되지 않은 여러 가지 개념을 소개하고 있다.
9) temporal databases를 모델링하기 위하여, 수많은 ER 개량형에 대한 고려가 이루어졌다. 유사하게도, ER 모델은 OLAP 어플에서 사용된 다차원적인 데이터베이스에서는 부적당한 것으로 나타났다: 비록 이것들이 일반적으로 OLAP cube의 개념(이 분야에서는 data cube로 알려진)을 재사용하고 있다 하더라도, 어떠한 지배적인 개념적 모델도 아직 이 분야에서는 출현하지 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.

5) Enhanced entity-relationship model

EER 모델은 ER 모델에서 소개된 모든 개념을 포함하고 있다. 추가적으로 이것에는 세분화와 일반화(specialization and generalization)의 개념으로 subclass와 superclass(Is-a)의 개념을 포함하고 있다. 더구나, 이것은 서로 다른 객체 유형의 사물 union인 사물 집단을 표현하는데 사용되는 union type이나 category의 개념을 도입하고 있다.

5-1) Subclass and superclass

만일 또는 단지 모든 Y가 반드시 하나의 X를 갖는다면, 객체유형 Y는 객체유형 X의 하위 유형(하위클라스)이다. 하위 클라스 객체는 그것의 상위 클라스 객체의 모든 속성과 관계를 포함하고 있다. 하위 클라스 객체는 자체적으로 특수한 속성과 관계를 가질 수 있다(모든 속성과 관계는 모두 다 그것의 상위 클래스로부터 물려받는다). 가장 일반적인 상위 클라스의 한 가지는 하위 클라스로 승용차와 화물차를 포함하는 자동차이다. 승용차와 화물차 간에는 수많은 공동의 속성이 있으며, 비록 상위 클라스의 일부가 될 수도 있지만 승용차와 화물차에 한정된 속성(최대적재량, 화물차 유형 등)은 두 가지의 하위 클라스에서 표현한다.

6) Object model

사물 데이터베이스 또는 사물-지향형 데이터베이스 관리 시스템이란 사물 지향형 프로그래밍에서 사용된 것처럼, 사물의 형태로 정보를 표현하는 데이터베이스 관리 시스템이다. 사물 데이터베이스는 테이블 지향적인 관계형 데이터베이스와 다르다. 사물-관계형 데이터베이스는 이 두 가지 방법의 혼합형이다.

사물 지향형 데이터베이스 관리 시스템인 OODBMSs는 데이터베이스 기능과 사물-지향형 프로그래밍 언어 기능이 결합된 것이다. OODBMSs는 사물-지향형 프로그래머로 하여금 제품을 개발하고 사물처럼 그것을 저장하며 OODBMS에서 새로운 사물을 만들기 위하여 기존의 사물을 복사하거나 변경한다. 이 데이터베이스는 프로그래밍 언어와 통합되어 있기 때문에, 하나의 환경에서 프로그래머는 OODBMS와 프로그래밍 언어 둘 다를 표현하는데 있어서 똑같은 모델을 사용할 수 있으므로 일관성을 유지할 수 있다. 이와는 대조적으로 관계형 DBMSs 프로젝트는 데이터베이스 모델과 그 어플간에 보다 명확한 구분이 이루어져 유지된다.

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

7) Document model

다큐멘트-지향적 데이터베이스는 semi-structured data로 알려진 도큐멘트-지향적 정보를 저장, 검색, 관리하기 위해 디자인된 컴퓨터 프로그램이다. 이 데이터베이스는 소위 NoSQL 데이터베이스의 주요 범주 중의 하나이며, “다큐멘트-지향적 데이터베이스” 또는 “document store”라는 용어의 인기가 NoSQL 용어와 더불어 사용이 증가하고 있다. 관계형 데이터베이스와 그것의 “관계” 또는 “테이블”의 개념과는 대조적으로, 이 시스템들은 “다큐멘트”의 추상적 개념을 가지고 디자인한다.

1) 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).

다큐멘트 지향적 데이터베이스의 중심 개념은 다큐멘트의 개념이다. 각각의 다큐멘트-지향적 데이터베이스를 실행

하는데 있어서는 일반적으로 이러한 정의의 내용과는 차이가 나지만, 이것들은 모두 다 다큐멘트가 어떤 표준 포맷이나 암호화를 통해 데이터나 정보를 암호화하는 것으로 여겨지고 있다. 사용가능한 암호화에는 XML, YAML, JSON, and BSON 뿐만 아니라 PDF와 Microsoft Office 다큐멘트와 같은 이진 형태(MS Word, Excell 등)가 있다.

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.

다큐멘트-지향적 데이터베이스 내부의 다큐멘트는 몇 가지 방법에서 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 다큐멘트이다.

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

A second document might be:

두 번째 다큐멘트는 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 다큐멘트는 서로서로 몇 가지의 구조적 요소를 공유하고 있지만, 각각은 또한 유일한 요소들을 가지고 있다. 사용되지 않는 필드는 빈 공간으로 남겨 놓음으로써, 모든 레코드가 동일한 필드를 갖고 있는 관계형 데이터베이스와 달리, 이것에는 어떠한 빈 공간의 “필드”도 위의 예에서처럼 존재하지 않는다. 이러한 방법으로 새로운 정보는 그 데이터베이스의 모든 다른 레코드가 동일한 구조를 공유할 필요 없이, 어떤 레코드에 추가될 수 있다.

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.

다큐멘트는 그 다큐멘트를 대표하는 유일한 키를 통해 데이터베이스에 자리를 잡는다. 이 키는 종종 간단한 문자열, URI, 또는 path 이다. 이 키는 데이터베이스로부터 그 다큐멘트를 검색하는데 사용될 수 있다. 전형적으로, 이 데이터베이스는 다큐멘트 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.

다큐멘트-지향형 데이터베이스의 또 다른 분명한 특징은 다큐멘트의 검색에 사용할 수 있는 간단한 key-document(or key-value) lookup 이외에도, 그 데이터베이스는 사용자로 하여금 콘텐트를 근거로 다큐멘트를 검색하도록 하는 API나 쿼리 언어를 제공한다는 것이다. 예를 들어, 여러분이 어떤 값으로 어떤 필드 세트를 가지고 있는 모든 다큐멘트를 검색할 수 있는 쿼리를 원할 수 있다. 이용가능한 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 다큐멘트를 조직하는 다양한 방법이 실행 된다.

- Collections
- Tags
- Non-visible Metadata
- Directory hierarchies
- Buckets

8) Entity-attribute-value model

EAV 모델은 객체를 기술하는데 사용할 수 있는 속성(성질, 변수)의 수가 잠재적으로 방대하지만, 실제적으로 특정한 객체에 적용할 수 있는 숫자가 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 schema로 부르기도 한다.

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 여겨고 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 기술 된다: “long”은 로우의 수를, 그리고 “skinny”는 소수의 칼럼을 의미한다.

Data is recorded as three columns:
데이터는 3개의 칼럼에 저장된다:

- The *entity*: the item being described.: *객체*: 기술대상의 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 칼럼이 포함될 수 있다: 속성 ID, 속성 이름, 정의, 데이터 유형, 그리고 입력의 타당성을 지원하는 칼럼, 예를 들어, 최대의 문자열 길이와 정상적인 표현, 허용가능한 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(폐렴).

(<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(담), yellowish, streaks(줄, 선) 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.

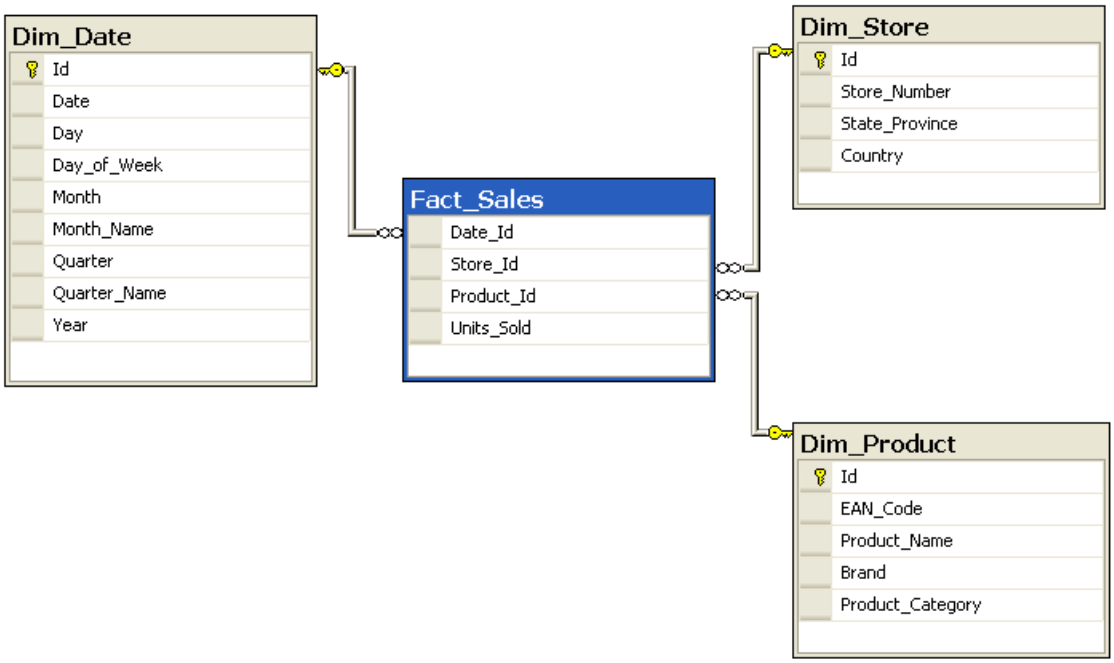
- As noted above, EAV modeling makes sense for categories of data, such as clinical findings, where attributes are numerous and sparse(성긴). Where these conditions do not hold, standard relational modeling (i.e., one column per attribute) is preferable; using EAV does not mean abandoning common sense 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

스타 스키마(star-join schema라고도 부른다)는 data mart 스키마의 가장 간단한 유형이다. 스타 스키마는 몇 개의 dimension tables을 참조하는 하나 이상의 fact tables로 구성된다. 스타 스키마는 snowflake 스키마의 중요하고 특별한 경우이며, 보다 간단한 쿼리를 처리하는데 있어서 보다 효과적이다.

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
```

물리적 데이터 모델에는 다음과 같은 것이 포함된다:

a) Inverted index

도치색인(또는 **postings file** or **inverted file**)은 단어와 숫자와 같은 콘텐트에서부터 데이터베이스 파일 또는 한 다큐멘트나 다큐멘트의 세트에 있는 그것의 위치까지 추적할 수 있는 mapping을 저장하는 색인 데이터 구조이다. 역색인의 목적은 다큐멘트가 그 데이터베이스에 추가될 때 처리비용이 늘어나더라도 신속한 전문탐색이 가능하도록 하는 것이다. 도치파일은 색인이라기보다는 데이터베이스의 그 자체 파일일 수 있다. 이것은 탐색엔진에서 많은 예를 들고 있는 다큐멘트 검색 시스템에서 사용되는 가장 인기 있는 데이터 구조이다. 여러 가지의 중요한 범용의 대형 컴퓨터용 데이터베이스 관리 시스템인 ADABAS, DATACOM/DB, Model 204 등에서 도치 리스트 구조를 사용하고 있다.

도치색인에는 두 가지 중요한 유형이 있다:

a) record level inverted index(또는 inverted file index 또는 단지 inverted file)은 각 단어에 맞는 다큐멘트들에 대한 참고 리스트를 가지고 있다.
b) word level inverted index(또는 full inverted index 또는 inverted list)는 추가적으로 다큐멘트내의 각 단어의 위치를 포함하고 있다.
후자는 더 많은 기능성(어구 탐색과 같은)을 제공하지만 만드는데 시간과 공간이 더 많이 필요하다.

<예> 다음과 같은 텍스가 주어졌을 때,:

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

우리는 다음과 같은 inverted file index을 갖고 있다(the set notation brackets 속에 있는 정수들은 텍스트 심볼인 T[0], T[1] 등)의 색인(또는 키)를 의미한다).

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

“what”, “is”, “it”의 용어 탐색을 하면 다음과 같은 집합을 얻을 수 있다.

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

똑같은 텍스트와 함께, 우리는 각각의 쌍이 다큐멘트 번호와 해당 단어 번호인 다음과 같

은 full inverted index을 얻게 된다. 다큐멘트 번호와 마찬가지로 해당 단어 번호 역시 0으로 시작하므로, "banana":{[2,3]}은 단어"banana"가 3번째 도큐멘트(T[2])에 있으며, 이것은 그 다큐멘트에서 4번째에 위치한 단어라는 것을 의미한다(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)**}

우리가 어구탐색 "what is it"을 실행한다면, 우리는 다큐멘트 0과 1 둘 다로부터 모든 단어를 hits시킬 수 있다.

<Applications>

도치색인 데이터 구조는 전형적인 탐색엔진 색인 알고리즘의 핵심적인 구성요소이다. 탐색엔진의 실행 목적은 쿼리의 속도를 최적화시키는 것이다: 즉, 단어 X가 발생하는 다큐멘트를 찾는 것이다. 일단 다큐멘트 당 단어의 리스트를 저장하고 있는 forward index이 개발되었다면, 그 다음은 도치색인을 개발하기 위하여 그것을 도치시키는 것이다. forward 색인에 쿼리하기 위해서는 매칭되는 다큐멘트를 밝히기 위하여 각 다큐멘트에 있는 각각의 단어를 순차적 반복할 필요가 있다. 그러나 그 같은 쿼리를 수행하기 위해 필요한 시간, 메모리, 그리고 처리 자원들이 항상 기술적으로 실현되지는 않는다. forward 색인에 있는 다큐멘트 당 단어를 리스트하는 대신에, 도치 색인 데이터 구조는 단어 당 다큐멘트를 리스트하도록 개발되었다.

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.

b) Flat file

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. 그렇지만 대체로 레코드들 간에는 어떠한 구조적 관계도 갖고 있지 않다.

<Overview>

plain text files은 대체로 줄 당 한 개의 레코드를 가지고 있으며, 데이터를 표현하기 위한 여러 가지 규정이 있다. comma-separated values(CSV) 와 delimiter-separated values files에 있어서 필드들은 comma나 tab character와 같은 delimiters에 의해 분리될 수 있다. 또다른 경우에, 각 필드는 고정장일 수도 있다: 짧은 값들은 공간(space) 문자와 더불어 첨부될 수 있다. extra formatting이 delimiter collision을 피하기 위하여 필요할 때도 있으며, 보다 복잡한

solutions으로는 markup language와 programming languages가 있다.

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.

delimiters를 사용하는 것은 그것들을 처리(고정장 길이의 포매팅과는 달리)할 때마다, 그것의 위치를 결정해야 하는 어느 정도의 부담(overhaead)이 발생한다. 그렇지만 character delimiters(특히 쉼표)를 사용하는 것은 데이터의 크기를 줄임으로써 - 특히 데이터 전송을 목적으로 하는 경우에 - 전반적인 성능에 도움을 주는 데이터 압축의 원형이다. a length component (Declarative notation)를 포함하고 있는 character delimiters의 사용은 비교적 드물지만 각 필드의 범위로 인하여 발생하는 부담을 크게 줄인다.

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

a sheet of paper에 직접 기록한 names, addresses, phone numbers의 리스트는 일종의 flat file 데이터베이스이다. 이것은 또한 타자기나 워드프로세서에 의해 작성될 수 있다. 스프레드쉬트나 텍스트 편집 프로그램은 플랫파일 데이터베이스를 기동시키기 위하여 사용될 수 있다. 그런 다음에 출력할 수도 있고 또는 개선된 탐색성능을 가지고 온라인으로 사용할 수도 있다.

다음의 예는 플랫 파일 데이터베이스의 기본적 요소를 나타내고 있다. 데이터 배치(arrangement)은 테이블 포맷으로 조직된 일련의 칼럼과 로우로 이루어져 있다. 이 특별한 예에서는 단지 하나의 테이블만을 사용한다. 칼럼들에는 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)를 사용하고 있다.

<예>

id	name	team
1	Amy	Blues
2	Bob	Reds
3	Chuck	Blues
4	Dick	Blues
5	Ethel	Reds

6	Fred	Blues
7	Gilly	Blues
8	Hank	Reds

비록 텍스트로 표현하는 것이 분명히 쉽지 않다는 몇 가지 추가적인 고려사항이 있더라도, 이러한 유형의 데이터 표현이 플랫-파일 데이터베이스용으로는 적합한 기준이다.

- **Data types:** 데이터 유형: 데이터베이스 테이블에 있는 각 칼럼은 보통은 특별한 데이터 유형으로 제한되어 있다. 그 같은 제한은 대체로 규정에서 마련되지만, 만일 그 데이터가 관계형 데이터베이스 시스템으로 전해지지 않는다면 공식적으로 지정되지는 않는다.

- **Separated columns:** 분리된 칼럼: 위의 예에서처럼, 각 칼럼들은 whitespace 문자를 사용하여 분리되어
  있다. 이것을 indentation 또는 "fixed-width" data formatting이라고도 부른다.
  또 다른 일반적인 규정은 하나 이상의 delimiter 문자를 사용하여 칼럼을 분리 하는다. 보다 복잡한 해결책으로는 Markup과 programming language를 사용하는 것이다.

- **Relational algebra:** 관계 대수: 위의 테이블에 있는 각각의 로우나 레코드는 관계형 대수의 tuple의 표 준 정의를 충족시키고 있다(위의 예에서는 3-tuples로 표현되어 있다). 따라서, 첫 번째 로우는 각 로우의 값들과 결합되어있는 name 필드를 나타내고 있다.
  관계형 대수의 주요 어플은 관계형 데이터베이스, 특히 그 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:>>

데이터베이스 관리 시스템: 텍스트 파일과 함께 공식적으로 운영할 수 있는 가능성이 대체로 원하는 것보다 더욱 제한됨으로써, 위의 예에 있는 텍스트는 보통 데이터베이스 관리 시스템에 전달되기 전에 나타나는 데이터의 중간매체 상태를 나타낸다.

기타 모델:

a) Associative model

데이터의 조합식 모델은 데이터베이스 시스템용의 대안적 데이터 모델이다. 관계형 모델과 사물 데이터 모델과 같은 데이터 모델들은 레코드 의존적이다. 이러한 모델들은 레코드 구조 속에서 자동차와 같은 하나의 사물에 대한 속성들을 갖고 있다. 이 같은 속성들은 등록증, 색깔, 제조, 모델 등일 수도 있다. 조합식 모델에서 "분명한 독립적 존재"의 모든 것은 하나의 사물처럼 모델화 되며, 그것들 간의 관계는 조합(association)처럼 모델화 된다. 또한 데이터를 표현하는 구조(granularity)는 Chen(객체-관계 모델); Bracchi, Paolini와 Pelagatti(Binery Realtions); Senko(The Entity Set Model)에서 제시한 스키마와 비슷하다.

b) Multidimensional model

다차원 구조는 "데이터를 조직하고 데이터 간의 관계를 표현하는데 있어서 다차원 구조를 사용하는 관계형 모델의 한 변종"으로 정의되고 있다. 그 구조는 cubes(입방체)로 나누어지고, 그 cubes는 각각 자신의 범위 내에서 데이터를 저장하고 접근할 수 있다. "다차원 구조에 있는 각 셀은 각각의 차원에 따라 존재하는 요소들과 관련되어 수집된 데이터를 포함하고 있다." 데이터를 조작할 때조차도, 압축 데이터베이스 포맷에 접근하기 편하게 지속적으로 그 구조를 유지한다. 따라서 그 데이터는 아직까지 상호연관성을 유지하고 있다. 이러한 다차원 구조는 online analytical processing(OLAP) 어플을 사용하는 분석적 데이터베이스에서 매우 인기가 높다.

1) The core of any **OLAP** system is an OLAP cube (also called a 'multidimensional 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.

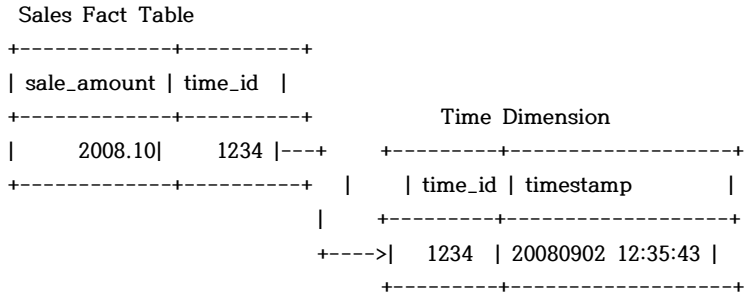
The cube metadata is typically created from a star schema or snowflake schema or fact constellation of tables in a relational database. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables.

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 Customer 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:



분석적 데이터베이스에서는 이러한 데이터베이스를 사용하는데 그 이유는 데이터를 다른 모델과 달리 어떤 문제를 보는데 보다 넓은 시각으로 볼 수 있도록 다양한 각도를 제공함으로써, 복잡한 경영상의 쿼리에 대하여 신속하게 해답을 제공할 수 있는 능력을 갖고 있기 때문이다.

c) Multivalue model

MultiValue는 NoSQL과 다차원 데이터베이스의 한 유형이며, Pick 운영체계용으로 원래 개발된 데이터베이스인 PICK와 전형적으로 동의어로 여겨지고 있다. MultiValue 데이터베이스는 Rocket Software, TigerLogic, jBASE, Revelation, Ladybridge, InterSystems, Northgate Information Solutions and other companies에서 만든 상업적 제품에서 사용하고 있다. 이 데이터베이스들은 단일 값으로 된 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 SQL과 상관없이 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.

d) Semantic model

씨멘틱 데이터 모델은 소프트웨어 공학에서 여러 가지의 의미를 가지고 있다:

1. 어의 정보를 포함하고 있는 개념적 데이터 모델인데, 이 뜻은 그것의 instances에 대한 의미를 기술하는 모델이란 것이다. 이러한 어의적 데이터 모델은 저장된 심볼(the instance data)이 실세계와 어떻게 관련되어 있는지를 설명하는 하나의 abstraction이다.

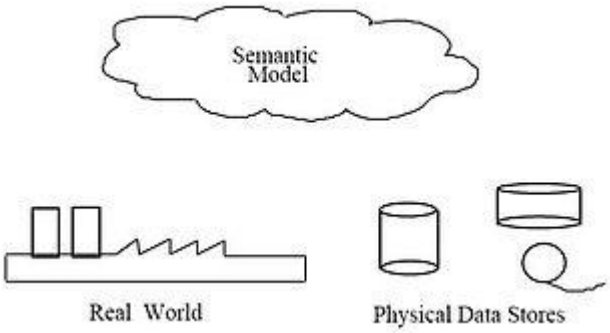
2. 이것은 Meta-model을 알 필요도 없이, instances로부터 의미(어의)를 해석할 수 있도록 parties로 하여금 정보교환이 가능하다는 정보를 표현할 능력을 갖고 있는 개념적 데이터 모델이다. 이러한 어의적 모델은 fact-oriented 이다(객체 지향적의 반대). 사실은 데이터 요소 간의 이진관계에 의해 전형적으로 표현된다. 반면에 고차원의 order 관계는 이진 관계의 집단으로 표현된다. 전형적으로 이진관계는 triples의 형태는 Object-Relation Type-Object로 이루어진다: 예) the Eiffel Tower <is located in> Paris.

전형적으로 말해서, 씨멘틱 데이터 모델의 instance data는 다양한 데이터 요소 간의 여러 종류의 관계(<is located in>처럼)를 분명하게 포함한다. 그러한 instances로부터 사실의 의미를 해석하기 위하여, 필요한 것은 관계의 종류(관계 유형)에 포함되어 있는 의미를 파악하는 것이다. 따라서 씨멘틱 데이터 모델에서는 전형적으로 그 같은 관계 유형을 표준으로 사용하고 있다. 이것은 2번째 유형의 씨멘틱 데이터 모델이 의미를 포함하고 있는 사실을 그것의 instance로 하여금 표현할 수 있다는 뜻이다. 2번째 종류의 씨멘틱 데이터 모델이란 대체로 씨멘틱 데이터베이스를 만드는 것을 의미다. 이러한 씨멘틱 데이터베이스의 의미 포함 능력은 어플로 하여금 콘텐트로부터 의미를 해석할 수 있는 분산형 데이터베이스를 구축하는 것을 용이하게 한다. 이 말은 씨멘틱 데이터베이스는 그것들이 동일한 또는 표준화된 관계 유형을 사용할 때 통합될 수 있다는 뜻이다. 그리고 또한 이것은 일반적으로 이러한 데이터베이스가 관계형 또는 객체 지향형 데이터베이스보다 더욱 다양한 응용성을 가지고 있다는 뜻이기도 하다.

<Overview>

DBMS의 논리적 데이터 구조가 계층적이든, 네트워크든, 또는 관계형이든 상관없이 데이터의 개념적 정의에 필요한 모든 것을 만족시키지는 못하는데, 왜냐하면 범위가 제한되어 있고 그 DBMS의 실행 전략에 편향되어 있기 때문이다. 그러므로 개념적 view로부터 데이터를 정의하려는 필요성이 씨멘틱 데이터 모델링 기법 즉, 다른 데이터와의 상호연관성을 고려하여 데이터의 의미를 정의하는 기법을 발전시켰다. 아래 그림에서처럼, 자원, 아이디어, 이벤트 등과 관련된 실세계는 상징적으로 물리적 데이터 저장고에서 정의된다. 씨멘틱 데이터 모델은 그 저장된 심볼이 어떻게 실세계와 관련되어 있는지를 정의하는 추상이다. 그러므로 그 모델은 실세계를 진솔하게 표현해야만 한다.

Klas와 Schrefl(1996)에 따라, "씨멘틱 데이터 모델의 전체 목표는 관계적 개념과 더불어 인공지능분야에서부터 알려진, 보다 강력한 추상 개념을 통합함으로써 데이터에 관한 더 많은 의미를 수집하는 것이다. 이런 아이디어는 실세계의 표현을 원활하게 하기 위하여 데이터 모델의 필수적 요소로서 high level modeling primitives를 제공하고 있다."



e) XML database

XML 데이터베이스는 데이터를 XML 포맷에 저장할 수 있는 data persistence software system 이다. 이렇게 저장된 데이터는 원하는 포맷으로 queried, exported, 그리고 serialized될 수 있다. XML 데이터베이스는 대체로 다큐멘트 지향적 데이터베이스와 결합한다.

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.

중요한 두 가지 유형의 XML 데이터베이스:

1. XML-enabled: 이 데이터베이스는 입력으로 XML을 받거나 출력으로 XML을 보내는 전통적인 데이터베이스 구조(관계형 데이터베이스처럼)를 XML로 작성(map)하거나, 또는 최근들어, 전통적인 데이터베이스에서 native XML 유형을 지원하도록 한다. 그리고 이 용어가 암시하는 것은 데이터베이스가 스스로 XML을 처리한다(middleware에 의존하는 것과는 반대) 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): 이 데이터베이스의 내부 모델은 XML에 의존하며 기본적인 저장 단위로 XML 다큐멘트를 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
3.XML is stored into a native XML Type as defined by the ISO

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

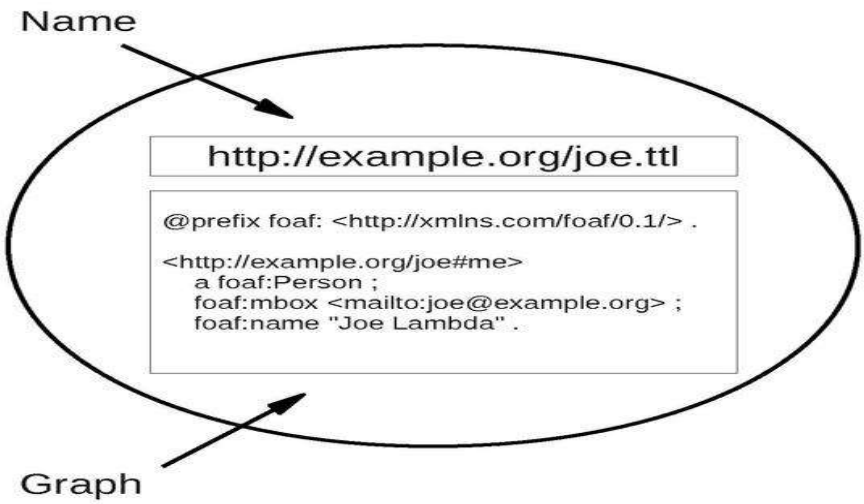
them support XPath for performing queries against documents or collections of documents. XPath provides a simple pathing system that allows users to identify nodes that match a particular set of criteria. In addition to XPath, many XML databases support XSLT as a method of transforming documents or query-results 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.

f) Named graph

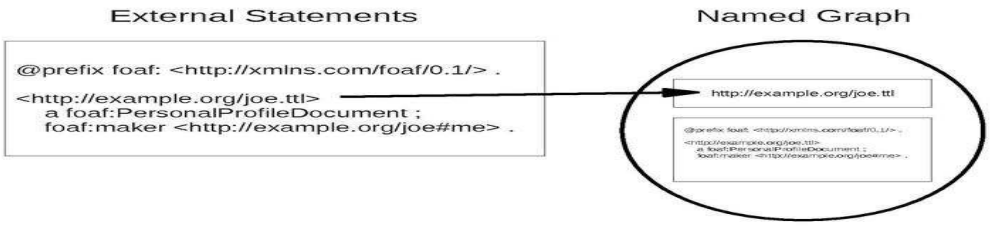
Named graphs는 한 세트의 Resource Description Framework statements(a graph)를 URI를 사용하여 식별하는 Semantic Web 구조의 중요한 개념이다. 여기서 URI는 context, provenance(기원, 출처) 정보, 또는 기타 메타데이터와 같은 statements로 구성된 이러한 세트를 만들도록 하는 description을 허용하고 있다. Named graphs는 graph를 제작할 수 있는 RDF 데이터 모델을 간단하게 확장시킨 것이지만, 이 모델은 일반적으로 일단 웹으로 출판되면 그것들을 구별하는 효과적 수단으로는 부족하다.



<Named graphs and HTTP>

웹의 한 가지 개념화는 URIs로 식별된, 그리고 HTML 다큐멘트에서 표현된 hyperlink

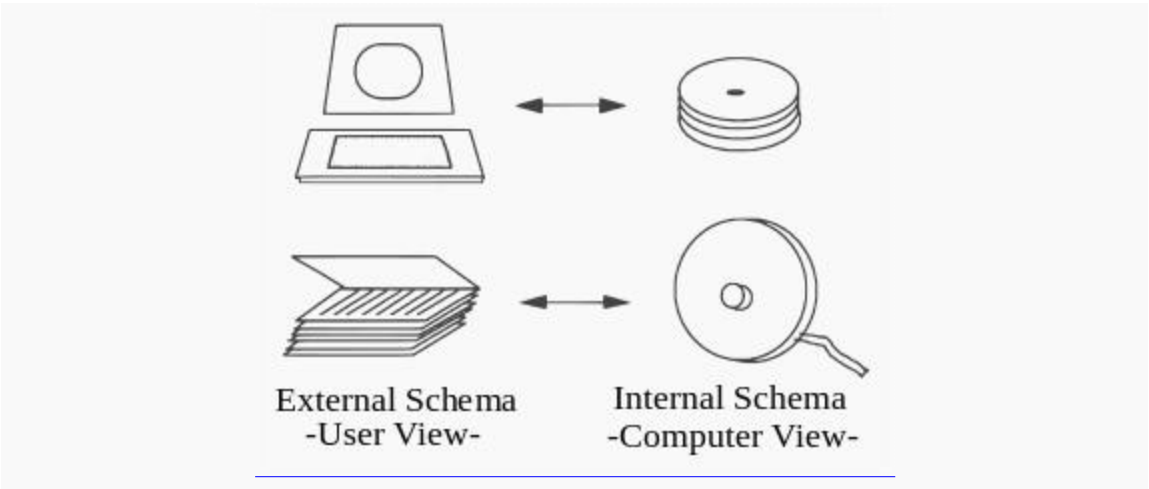
arc로 연결된 다큐멘트 nodes의 그래프이다. (대체로 웹 브라우저를 통해) URI에서 HTTP GET를 실행함으로써, 어느 정도(somehow)-연관된 다큐멘트가 검색될 수 있다. 이러한 "follow your nose" 어프로치 또한 Linked Data(전형적으로 RDF syntax가 statements 시리즈처럼 데이터를 표현하는데 사용되는 형태)와 다른 자원으로 가기 위하여 RDF point 속에 있는 URIs의 형태로 웹에 있는 RDF 다큐멘트에 적용된다. 데이터에 대한 이 같은 웹은 Time Berners-Lee에 의해 "Giant Global Graph"로 기술되고 있다.



Describing a named graph

Named graphs는 웹의 RDF 다큐멘트(a graph)의 콘텐트가 그 다큐멘트의 URI에 의해 명명될 수 있다는 직관적 아이디어를 공식화한 것이다. 이것은 상당히 데이터의 출처 고리(chains of provenance)를 관리하고 그 source data로의 접근을 효과적으로 결과를 얻도록(fine-grained) 통제할 수 있는 기술을 단순화시킨 것이다. 추가적으로 named graph에 있는 데이터에 디지털 서명을 applying하고 있는 출판사를 통하여 신뢰가 어느 정도 이루어지고 있다(이러한 기능의 지원은 원리 RDF reification(구상화) 단계에서부터 하려고 했으나 그러한 시도가 문제가 있는 것으로 나타났다).

6.2 External, conceptual, and internal views



Traditional view of data

데이터베이스 관리 시스템은 데이터베이스 데이터에 대하여 3가지의 views를 제공한다:

1) The external level
   외적 차원은 엔드유저의 각 그룹이 데이터베이스에 있는 데이터의 조직을 아는 방법을 정의한다. 단일 데이터베이스는 외적 차원에서 어느 정도의 views를 가질 수 있다.
2) The conceptual level
   개념적 차원에서는 다양한 외적 뷰를 호환 가능하고 보편적인 뷰로 통합한다. 따라서 이것은 모든 외적 뷰의 통합을 제공한다. 또한 이것은 다양한 데이터베이스 엔드유저의 범위가 아니라 그것보다는 데이터베이스 개발자와 데이터베이스 행정가의 관심 대상이다.
3) The internal level (or *physical level*)
   내적 차원(또는 물리적 차원)은 DBMS에 있는 데이터의 내적 조직을 말한다. 이것은 비용, 성능, 확장성(scalability), 그리고 기타 운영업무와 관련이 있다. 이것은 성능을 높이기 위하여 색인과 같은 저장 구조를 사용함으로써 데이터의 저장 모습을 다룬다. 경우에 따라서, 만일 성능 확인이 잉여정보에 대하여 필요하다면, 이것은 포괄적(generic) 데이터로부터 계산된 각각의 뷰(물질화된 견해들)에 대한 데이터를 저장한다. 또한 이것은 모든 활동에 대하여 전체적인 성능을 최적화하려는 시도에 따라 가능한 한 부딪쳐 보면서(conflicting) 모든 외적 뷰의 성능조건과 균형을 맞추어야 한다.

전형적으로 데이터에 대한 한가지의 개념적(또는 논리적) 그리고 물리적(또는 내적) 뷰만 존재한다 하더라도, 어느 정도의 서로 다른 외적 뷰 역시 존재한다. 이것은 이용자로 하여금 기술적이고 처리 위주의 관점에서부터 더욱 업무-관련 중심의 방식으로 데이터베이스 정보를 이해할 수 있도록 한다. 예를 들어, 회사의 재무부는 회사 경비의 일부로서 모든 직원에 대한 급료 내역을 필요로 한다. 그러나 인사부에서 관심이 있는 직원에 대한 상세정보는 필요하지 않다. 그러므로 서로 다른 부서들은 그 회사 데이터베이스에서 서로 다른 views를 필요로 한다.

이러한 3 차원의 데이터베이스 구조는 관계형 모델의 중요한 기본 동력의 하나인 데이터 독립성과 관련이 있다. 이 아이디어는 어느 수준에서 일어난 변화는 그 보다 고차원에 있는 뷰에는 영향을 끼치지 않는다는 것이다. 예를 들어, 내적 차원의 변화는 개념적 차원의 인터페이스를 사용함으로써 기존의 응용 프로그램에 영향을 끼치지 않으므로, 성능 향상을 위한 물리적 변경 시도의 충동을 감소시킨다.

개념적 뷰는 내적 그리고 외적 간의 indirection(간접적 수단)의 수준을 제공한다. 한편으로 이것은 다양한 외적 뷰의 구조와는 독립적인 데이터베이스의 일반 뷰를 제공한다. 또한편으로 이것은 데이터가 어떻게 저장, 관리되는지에 대한 것(내적 수준)과는 거리가 멀다(abstracts away). 원칙적으로 모든 차원 그리고 심지어 모든 외적 뷰는 하나의 다양한 데이터 모델로 표현될 수 있다. 실재적으로, 대부분의 특정한 DBMS는 외적 그리고 개념적 차원용으로 동일한 데이터 모델을 사용하고 있다(예, 관계형 모델). DBMS에 숨어서 그것의 실행에 의존하는 내적 차원은 다양한 차원의 상세한 내용을 필요로 하며, 여러 데이터 구조 유형 중에서 자신의 고유한 유형을 사용한다.

외적 차원과는 달리, 개념적 그리고 내적 차원은 21세기 데이터베이스를 지배하고 있는 관계형 데이터베이스 모델의 실행에서 중요한 특징들이다.

7. Database languages

데이터베이스 언어는 특별한 목적의 언어이며 다음과 같다:

- Data definition language: 데이터 정의어 - 데이터 유형과 그것들간의 관계를 정의한다.
- Data manipulation language: 데이터 조작어 - 데이터의 입력, 갱신, 또는 삭제와 같은 업무를 수행한다.
- Query language: 쿼리어 - 정보의 탐색과 유도된 정보를 계산하도록 한다.

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

1) SQL

SQL은 단일 언어로 데이터 정의, 데이터 조작, 그리고 쿼리의 역할을 결합한 것이다. 이것은 비록 어떤 점에서는 Codd가 설명한 관계형 모델(예를 들어 테이블의 컬럼과 로우는 순서에 맞춰질 수 있다)과는 동떨어져 있다하더라도, 관계형 모델용의 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 순응의 정도(degrees of conformance)는 서로 다르더라도 지지를 얻고 있다.

2) OQL

OQL은 Object Data Management Group에서 만든 객체모델 언어 표준이다. 이것은 JDOQL과 EJB QL과 같이 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.

3) XQuery

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

4) SQL/XML

SQL/XML은 XQery와 SQL이 결합된 것이다.

데이터베이스 언어는 또한 다음과 같은 특징을 가질 수도 있다:

- DBMS-specific Configuration and storage engine management

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

7.1 SQL이란?

SQL(Structured Query Language)이란 relational database management system (RDBMS)에 포함된 데이터를 관리하도록 디자인된 또는 relational data stream management system (RDSMS)에서 stream processing를 위한 특별한 목적의 프로그래밍 언어이다.

원래는 relational algebra and tuple relational calculus를 근거로, SQL은 데이터 정의어와 데이터 조작어(data definition language and a data manipulation language)로 구성되어 있다. SQl의 범위에는 데이터의 입력, 쿼리, 갱신, 삭제 스키마 제작 및 변경 그리고 데이터 접근 통제가 포함된다. 비록 SQL이 어느 정도까지는 종종 declarative language(4GL)라 하더라도, 여기에는 또한 절차적 요소들이 포함되어 있다.

SQL은 Edgar F. Codd's relational model – in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks." –을 위한 최초의 상업적 언어들 중의 하나이다. Codd가 설명했듯이 관계형 모델에 전적으로 의존하지 않음에도 불구하고, 이것은 가장 널리 사용되는 데이터베이스 언어가 되었다.

SQL은 1986년에 the American National Standards Institute (ANSI), 그리고 1987년에 the International Organization for Standardization (ISO)의 표준이 되었다. 그 이후로, 이 표준은 많은 부분에서 수정되었다. 이 같은 표준이 존재함에도 불구하고, 대부분의 SQL 코드는 조정을 거치지 않는다면 서로 다른 데이터베이스 시스템 간에 완벽하게 이식(portable)될 수는 없다.

7.2 History

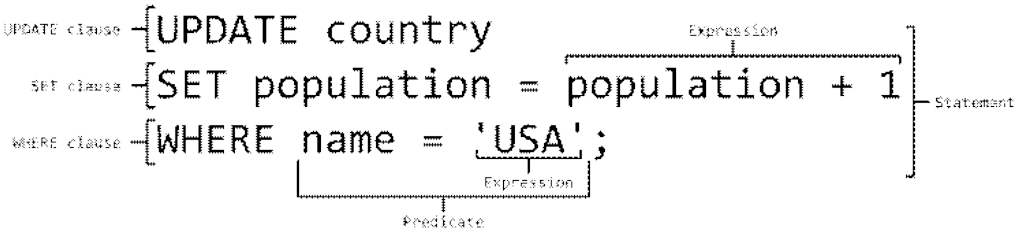
SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. This version, initially called SEQUEL (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's original quasi-relational database management system, System R, which a group at IBM San Jose Research Laboratory had developed during the 1970s. The acronym SEQUEL was later changed to SQL because "SEQUEL" was a trademark of the UK-based Hawker Siddeley aircraft company.

In the late 1970s, Relational Software, Inc. (now Oracle Corporation) saw the potential of the concepts described by Codd, Chamberlin, and Boyce, and developed their own SQL-based RDBMS with aspirations of selling it to the U.S. Navy, Central

Intelligence Agency, and other U.S. government agencies. In June 1979, Relational Software, Inc. introduced the first commercially available implementation of SQL, Oracle V2 (Version2) for VAX computers.

After testing SQL at customer test sites to determine the usefulness and practicality of the system, IBM began developing commercial products based on their System R prototype including System/38, SQL/DS, and DB2, which were commercially available in 1979, 1981, and 1983, respectively.

7.3 Syntax



7.3.1 Language elements

SQL 언어는 다음과 같이 여러 가지의 언어요소들로 세분된다:

1) Clauses; statements and queries를 구성하는 요소들이며, 경우에 따라서는 선택적이다.
2) Expressions; 데이터의 칼럼과 로우를 구성하는 테이블이나 scalar values를 생산할 수 있다.
3) Predicates; SQL three-valued logic (3VL) (true/false/unknown) 또는 Boolean truth values을 위해 평가될 수 있는 조건을 특정화 하며 statements and queries의 결과를 제한하거나 프로그램의 흐름을 바꾸기 위하여 사용된다.
4) Queries; 특정한 기준에 따라 데이터를 검색한다. 이것은 SQL의 중요한 요소이다.
5) Statements; 스키마와 데이터에 대한 영구적인 효과를 가질 수 있으며, transactions, program flow, connections, sessions, or diagnostics를 제어할 수도 있다.
    a) SQL statements 또한 semicolon (";") statement terminator을 포함하고 있다. 비록 모든 프랫폼에서 필요로 하는 것은 아니더라도, 이것은 SQL 문법의 표준으로 정의되고 있다.
6) Insignificant whitespace는 가독성을 위하여 SQL 코드를 쉽게 포맷할 수 있도록, SQL statements와 queries에서 무시된다.

7.3.2 Operators

Operator	Description	Example
=	Equal to	Author = 'Alcott'
<>	Not equal to (many DBMSs accept != in addition	Dept <>'Sales'

	to <>)	
>	Greater than	Hire_Date >'2012-01-31'
<	Less than	Bonus <50000.00
>=	Greater than or equal	Dependents >= 2
<=	Less than or equal	Rate <= 0.05
BETWEEN	Between an inclusive range	Cost BETWEEN 100.00 AND 500.00
LIKE	Match a character pattern	First_Name LIKE 'Will%'
IN	Equal to one of multiple possible values	DeptCode IN (101, 103, 209)
IS *or* IS NOT	Compare to null (missing data)	Address IS NOT NULL
IS NOT DISTINCT FROM	Is equal to value or both are nulls (missing data)	Debt IS NOT DISTINCT FROM - Receivables
AS	Used to change a field name when viewing results	SELECT employee AS 'department1'

7.3.2.1 Conditional (CASE) expressions

SQL은 SQL-92에서 소개되었던 case/when/then/else/end와 같은 표현식을 갖는다. 이것의 가장 일반적인 형태는 SQL 표준에서 "searched case"라 부르며 기타 프로그램 언어에서 else if 처럼 사용된다:

```
CASE WHEN n > 0
         THEN 'positive'
     WHEN n < 0
         THEN 'negative'
     ELSE 'zero'
END
```

SQL은 소스네서 그것들이 나타나는 순서에 따라 WHEN 조건을 테스트한다. 만일 소스가 ELSE 표현을 특정화하지 않는다면, SQL은 ELSE NULL을 디폴트로 처리한다. SQL 표준에서 "simple case"라 부르는 단축 구문(abbreviated syntax)은 switch statements와 같다(mirrors):

```
CASE n WHEN 1
           THEN 'one'
       WHEN 2
           THEN 'two'
       ELSE 'I cannot count that high'
END
```

이 구문에서는 NULL과 비교하기 위하여 일상적인 caveats(정지통보, 보호신청, 경고, 단서)와 함께, implicit equality comparisons을 사용하고 있다.

Oracle-SQL dialect에서, 후자는 동일한 DECODE 구조로 단축될 수 있다:

```
SELECT DECODE(n, 1, 'one',
                 2, 'two',
                 'i cannot count that high')
```

FROM some_table;

마지막 값은 디폴트이다; 만일 어느 것도 특정화되지 않는다면, 그것 역시 NULL로 디폴트된다. 그렇지만 표준의 “simple case”와 달리, Oracle's DECODE는 서로 평등하게 두 개의 NULLs를 고려하고 있다.

7.3.3 Queries

SQL에서 가장 공통적인 기능은 쿼리이다. 이것은 declarative SELECT statement(명령문, 표현문)에서 이루어진다. SELECT는 하나 이상의 테이블 또는 표현에서 데이터를 검색한다. 표준 SELECT statements는 그 데이터베이스에 대하여 어떠한 항구적 효과도 갖지 않는다. SELECT의 어떤 비-표준적 실행에서는 몇몇 데이터베이스에 존재하는 SELECT INTO 구문처럼 항구적 효과를 가질 수도 있다.

쿼리들은 선택에 의해 결과를 생산하는데 필요한 물리적 기능을 기획하고, 최적화하고, 발휘하는데 책임이 있는 DBMS와는 별도로, 이용자로 하여금 필요한 데이터를 설명(describe)하도록 한다.

쿼리는 즉각적으로 SELECT 키워드를 따라감으로써 최종 결과에 포함되는 칼럼 리스트가 포함된다. asterisk ("*") 또한 그것의 쿼리가 쿼리되는 테이블들(queried tables)의 모든 칼럼을 return하야 한다는 것을 특정화하기 위하여 사용될 수 있다. SELECT는 SQL에서 그것에 포함되는 선택적 키워드와 절(clauses)과 더불어 가장 복장한 statement이다.

FROM 절은 데이터를 검색하기 위한 테이블을 지정한다. FROM 절은 테이블들을 결합(joining)하기 위한 규칙을 정하기 위하여 선택적으로 JOIN 하위절을 포함할 수 있다.

WHERE 절은 쿼리에 의해 나타난(returned) 로우들을 제한하는 비교 술어(comparison predicate)를 포함한다. WHERE 절은 비교술어가 참(True)이라고 평가하지 않는다면, 결과 세트로부터 모든 로우들을 제거한다.

GROUP BY 절은 공동의 값들을 가지고 있는 로우들을 보다 작은 세트의 로우들로 계획(project)하는데 사용된다. GROUP BY는 가끔 SQL aggregation(집합) functions와 결합해서, 또는 어떤 결과 세트로부터 중복된 로우들을 제거하기 위하여 사용되기도 한다. WHERE 절은 GROUP BY 절 앞에서 사용되어야 한다.

HAVING 절은 GROUP BY 절의 결과로부터 얻어지는 로우들을 거르는데 사용된 술어를 포함한다. 이것이 GROUP BY 절의 결과와 관현해서 행동하기 때문에, aggregation functions는 HAVING 절 술어 속에 사용될 수 있다.

ORDER BY 절은 어떤 칼럼들이 결과 데이터를 분류하는데 사용되고, 그것들을 어떤 방향(ascending or descending)에서 분류하는지를 사용되는지를 밝힌다. ORDER BY 절이 없다면, SQL에서 얻어진 로우들의 순서는 정의할 수 없다(undefined).

다음은 비싼 책들의 리스트를 보여주는 SELECT 퀄리의 예이다. 이 쿼리는 Book 테이블에서 price 칼럼이 100.00보다 큰 값을 갖고 있는 모든 로우들을 검색한다. 그 결과는 서명에 의한 상향식으로 분류된다. 선택 리스트에 있는 별표는 Book 테이블의 모든 칼럼들이 그 결과 세트에 포함되어야 한다는 것을 의미한다.

```
SELECT *
  FROM  Book
  WHERE price > 100.00
  ORDER BY title;
```

The example below demonstrates a query of multiple tables, grouping, and aggregation, by returning a list of books and the number of authors associated with each book.

아래의 예는 책의 리스트와 각 책의 저자의 수를 보여줌으로써 복수의 테이블, grouping, 그리고 aggregation의 쿼리를 보여주고 있다.

```
SELECT Book.title AS Title,
       COUNT(*) AS Authors
  FROM  Book
  JOIN  Book_author
    ON  Book.isbn = Book_author.isbn
  GROUP BY Book.title;
```

위의 예의 결과는 다음과 같을 수 있다:

```
Title                        Authors
---------------------------- --------
SQL Examples and Guide          4
The Joy of SQL                  1
An Introduction to SQL          2
Pitfalls of SQL                 1
```

isbn이 두 개의 테이블에서 오직 공동의 칼럼이라는 그리고 title이라고 이름 붙힌 한 칼럼이 Books 테이블에만 존재한다는 전제 하에서, 위의 쿼리는 다음과 같이 재 작성될 수 있다:

```
SELECT title,
       COUNT(*) AS Authors
 FROM  Book
 NATURAL JOIN Book_author
 GROUP BY title;
```

그렇지만, 많은 상인들이 이러한 시도를 지지하지도, 또는 효과적으로 작업하기 위하여 natural joins의 규정을 naming하는 어떤 칼럼을 요구하지 않고 있다.

SQL에는 저장된 값에 관한 값들을 계산하기 위한 연산자와 함수들이 포함되어 있다. SQL에서는 선택 리스트에 있는 표현식을 사용하여 아래의 예처럼 가격의 6%로 계산된 sales tax 숫자를 포함하고 있는 추가적 sales-tax 칼럼과 함께 100.00보다 비싼 값의 책 리스트를 보여주는 데이터를 나타내도록 한다(project).

```
SELECT isbn,
       title,
       price,
       price * 0.06 AS sales_tax
 FROM  Book
 WHERE price > 100.00
 ORDER BY title;
```

7.3.3.1 Subqueries

쿼리들은 한 개의 쿼리의 결과가 relational operator나 aggregation function를 통하여 또 다른 쿼리에서 사용될 수 있으므로 둥지화(nested)될 수 있다. 둥지화된 쿼리를 subquery라 부른다. joins와 다른 테이블 연산자들이 컴퓨터에서 많은 경우에 superior(다시 말해서, faser) 대안들을 제공하기 때문에, 하위 쿼리의 사용은 유용하거나 필요할 수 있는 실행에서 계층적으로 이루어진다. 다음의 예에서, aggregation function AVG는 하위 쿼리의 결과를 input으로 받는다:

```
SELECT isbn,
         title,
         price
 FROM  Book
 WHERE price < (SELECT AVG(price) FROM Book)
 ORDER BY title;
```

하위 쿼리는 바깥쪽(outer) 쿼리로부터 값을 사용할 수 있는데, 이런 경우를 correlated subquery라 부른다.

1999년 이래로 SQL 표준에서는 Common Table Expression을 요구하는 named subqueries를 허용하고 있다(Oracle calls these subquery factoring). CTEs는 또한 스스로를 참조함으로써 순환적일 있다; 최종결과의 메카니즘이 (관계들로 표현될 때) tree or graph traversals, 그리고 더욱 일반적인 fixpoint computations를 허용하고 있다.

7.3.3.2 Null and three-valued logic(3VL)

Null의 개념은 관계형 모델에서 결석한(missing) 정보를 다루기 위하여 SQL에서 소개되었다. 단어 NULL은 Null special marker를 식별하기 위하여 사용된 SQL의 예약 키워드이다. Null과 대조(Comparisons with Null)는, 예를 들어 WHERE 절에 있는 등식(=)처럼, Unknown이란 참 값을 결과로 가져온다. SELECT 명령문에서 SQL은 단지 WHERE 절이 True라는 값을

나타내는 결과만을 나타낸다; 다시 말해서, 그것은 False 값을 갖는 결과를 배제하며, 또한 그 값이 Unknown한 것들도 배제한다.

True and False에 따라, Null과의 직접적인 비교결과로부터 얻어지는 Unknown은 SQL을 위하여 한 조각(fragment)의 three-valued logic을 가져온다. 참 테이블 SQL은 a common fragment of the Kleene and Lukasiewicz three-valued logic (which differ in their definition of implication, however SQL defines no such operation)에 해당하는 AND, OR, and NOT용으로 사용된다.

p AND q		True	False	Unknown
q	True	True	False	Unknown
	False	False	False	False
	Unknown	Unknown	False	Unknown

p OR q		True	False	Unknown
q	True	True	True	True
	False	True	False	Unknown
	Unknown	True	Unknown	Unknown

True	False
False	True
Unknown	Unknown

p = q		True	False	Unknown
q	True	True	False	Unknown
	False	False	True	Unknown
	Unknown	Unknown	Unknown	Unknown

그렇지만 SQL에서 Nulls에 대한 어의적 해석에 대해서는 논란이 있는데, 그것의 처리가 직접적인 비교를 제외하기(outside) 때문이다. 위의 테이블에서 보듯이, SQL에 있는 두 가지의 NULLs 간의 직접적인 등식 비교(예, NULL=NULL)는 Unknown이라는 참 값을 보여준다. 이것은 Null이 값을 가질 순 없지만 그 보다는 빠진 정보에 대한 placeholder나 "mark"라는 것에 대한 해석에 따른 것이다. 그렇지만, 두 개의 Null들은 서로 똑같지 않아야 한다라는 원칙을 위반한 것인데, 그 이유는 nulls은 서로 식별되어야난 한다는 the UNION and INTERSECT 연산자에 대한 SQL 규정이 있기 때문이다. 결론적으로, SQL에서 NULL 함께 뚜렷한 비교가 가능한 연산자들과 달리(예, 위에서 논의한 WHERE 절에 있는 것들처럼), 이러한 조합 연산자들은 확실한(sure) 정보를 표현하지 못하는 결과를 생산할 수도 있다. (기본적으로 SQL92에서 채택된) 1979년 Codd의 제안에 있는 이러한 어의적 불일치는 조합연산자 속의 중복(duplicates) 제거가 검색 연산자의 평가를 테스트하는 equality 보다 더 낮은 수준의 detail(at a lower level

of detail than equality testing in the evaluation of retrieval operations)에서 발생한다는 주장을 합리화시키고 있다. 그렇지만, 컴퓨터학 교수 Ron von der Meyden이 결론내리길 "SQL 기준에서의 불일치가 의미하는 것은 SQL에서 nulls의 처리를 어떤 직관적인 논리적 어의(any intuitive logical semantics)에 기인할 수 없다"는 것이다. 추가로, SQL 연산자들이 어떤 것을 직접 Null과 비교할 때 Unknown을 나타내므로, SQL은 두 개의 Null-전용(specific) 비교 술어(데이터가 Null이냐 아니냐를 테스트하는 IS NULL과 IS NOT NULL)를 제공한다. Universal quantification을 SQL에서는 분명히 지원하지 않으며, a negated existential quantification처럼 이해(worked out)되어야만 한다. 또한 "<row value expression> IS DISTINCT FROM <row value expression>"처럼 (만일 두 개의 operand(연산 값)가 같지 않거나 두 개가 NULL이 아니라면 TRUE를 나타내는) infixed comparison operator(삽입 비교 연산자)도 존재한다. SQL:1999 또한 기준에 따라 또한 Unknown 값을 가질 수 있는 BOOLEAN 유형의 변수들을 소개하고 있다. 실재로 수많은 시스템(예, PostgreSQL)에서 BOOLEAN NULL로서 BOOLEAN Unknown을 사용하고 있다.

7.3.4 Data manipulation

데이터 조작어(Manipulation Language;DML)는 데이터를 추가, 갱신, 삭제하기 위하여 사용되는 SQL의 부분집합이다.

1) INSERT; rows (공식적으로는 tuples)를 기존 테이블에 추가, e.g.:

```
INSERT INTO example
 (field1, field2, field3)
 VALUES
 ('test', 'N', NULL);
```

2) UPDATE; a set of existing table rows를 변경, e.g.:

```
UPDATE example
 SET field1 = 'updated value'
 WHERE field2 = 'N';
```

3) DELETE; 테이블에서 기존 rows를 삭제, e.g.:

```
DELETE FROM example
 WHERE field2 = 'N';
```

4) MERGE; 복수 테이블의 데이터를 결합. 이것은 INSERT and UPDATE elements를 결합시킨다.

```
MERGE INTO TABLE_NAME USING table_reference ON (condition)
 WHEN MATCHED THEN
 UPDATE SET column1 = value1 [, column2 = value2 ...]
```

```
 WHEN NOT MATCHED THEN
 INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...])
```

7.3.5 Transaction controls

Transactions은 필요하면 DML 연산자들을 포장(wrap)할 수 있다operations:

1) START TRANSACTION (or BEGIN WORK, or BEGIN TRANSACTION, depending on SQL dialect): database transaction이 완료되었는지 또는 전혀 그렇지 않은지를 mark한다.
2) SAVE TRANSACTION (or SAVEPOINT): 거래의 현 시점에서 데이터베이스의 상태를 save한다.

```
CREATE TABLE tbl_1(id INT);
 INSERT INTO tbl_1(id) VALUES(1);
 INSERT INTO tbl_1(id) VALUES(2);
COMMIT;
 UPDATE tbl_1 SET id=200 WHERE id=1;
SAVEPOINT id_1upd;
 UPDATE tbl_1 SET id=1000 WHERE id=2;
ROLLBACK TO id_1upd;
 SELECT id FROM tbl_1;
```

3) COMMIT; 모든 데이터를 거래에서 항구적으로 변경하도록 한다.
4) ROLLBACK; 마지막 COMMIT or ROLLBACK 이후의 모든 데이터의 변화를 폐기하여, 데이터를 변경이전의 상태로 돌려놓는다. 일단 COMMIT statement가 완료되면, 그 transaction's 변경은 다시 되돌릴 수 없다.

COMMIT and ROLLBACK은 현재의 거래를 종료하고 데이터 잠금(locks)을 풀어놓는다. START TRANSACTION이나 비슷한 statement의 부재 시에, the semantics of SQL는 implementation-dependent 이다. 아래의 예에서 돈이 한 계좌에서 제거되어 다른 곳에 추가되는 자금 거래의 고전적 이동을 보여주고 있다. 만일 제거와 추가가 실패한다면, 모든 거래는 rolled back 된다.

```
START TRANSACTION;
 UPDATE Account SET amount=amount-200 WHERE account_number=1234;
 UPDATE Account SET amount=amount+200 WHERE account_number=2345;

IF ERRORS=0 COMMIT;
IF ERRORS<>0 ROLLBACK;
```

7.3.6 Data definition

데이터 정의어(Data Definition Language: DDL)는 table과 index structure를 관리한다. DDL의 가장 기본적인 아이템들은 CREATE, ALTER, RENAME, DROP and TRUNCATE

statements 이다:

1) CREATE ; 데이터베이스에 사물(예, a table)을 만든다, e.g.:

```
CREATE TABLE example(
 field1 INTEGER,
 field2 VARCHAR(50),
 field3 DATE NOT NULL,
 PRIMARY KEY (field1, field2)
);
```

2) ALTER; 기존 사물의 구조를 여러 가지 방식으로 변경한다(예, 기존 테이블에 칼럼이나 제한조건(constraint) 추가하기), e.g.:

```
ALTER TABLE example ADD field4 NUMBER(3) NOT NULL;
```

3) TRUNCATE; 매우 빠른 방식으로 테이블에서 모든 데이터를 삭제하지만, 테이블 그 자체는 삭제하지 않는다, 이것은 대체로 a subsequent COMMIT operation을 의미하는데, 다시 말해서, rolled back 될 수 없다는 것이다(데이터는 DELETE와 달리, 후에 rollback용 logs에 기록되지 않는다).

```
TRUNCATE TABLE example;
?
?
?
```

4) DROP; 데이터베이스에 있는 대체로 검색할 수 없는 사물을 delete 한다. 다시 말해서, 이것은 rolled back될 수 없다, e.g.:

```
DROP TABLE example;
?
?
?
```

7.3.7 Data types

SQL 테이블에 있는 각 칼럼은 소장이 가능한 유형을 선언하고 있다. ANSI SQL에서는 다음과 같은 데이터 유형을 갖고 있다.

1) Character strings

a) CHARACTER(n) or CHAR(n): fixed-width n-character string, padded with spaces as needed
b) CHARACTER VARYING(n) or VARCHAR(n): variable-width string with a maximum size of n characters
c) NATIONAL CHARACTER(n) or NCHAR(n): fixed width string supporting an international character set
d) NATIONAL CHARACTER VARYING(n) or NVARCHAR(n): variable-width NCHAR string

2) Bit strings
a) BIT(n): an array(배열, 배열된 데이터 군) of n bits
b) BIT VARYING(n): an array of up to n bits

3) Numbers
a) INTEGER, SMALLINT and BIGINT
b) FLOAT, REAL and DOUBLE PRECISION
c) NUMERIC(precision(정밀도), scale(척도, 진법, 배율, 축척)) or DECIMAL(precision, scale)

예를 들어, 번호 123.45는 5의 precision과 2의 scale를 갖고 있다. 정밀도란 (binary or decimal과 같이)특별한 radix(기수, 뿌리)에서 중요한 디지트의 숫자를 결정하는 실재적 정수(positive integer)를 말한다. 척도란 비-부정 정수(non-negative integer)를 말한다. 0 척도라는 것은 그 숫자가 정수라는 것을 의미한다. S 척도를 가진 십진 숫자에서 정확한 숫자 값은 10S로 나눈 의미 있는(significant) 디지트들의 정수 값이다.

SQL에서는 TRUNC (in Informix, DB2, PostgreSQL, Oracle and MySQL) or ROUND (in Informix, SQLite, Sybase, Oracle, PostgreSQL and Microsoft SQL Server)라 부르는 round (꼭 맞는) numerics나 dates를 위한 함수를 제공하고 있다.

4) Date and time
a) DATE: for date values (e.g. 2011-05-03)
b) TIME: for time values (e.g. 15:51:36). The granularity of the time value is usually a tick (100 nanoseconds).
c) TIME WITH TIME ZONE or TIMETZ: the same as TIME, but including details about the time zone in question.
d) TIMESTAMP: This is a DATE and a TIME put together in one variable (e.g. 2011-05-03 15:51:36).
e) TIMESTAMP WITH TIME ZONE or TIMESTAMPTZ: the same as TIMESTAMP, but including details about the time zone in question.

SQL에서는 date / time variable out of a date / time string (TO_DATE, TO_TIME, TO_TIMESTAMP)의 생산뿐만 아니라 그러한 변수들의 개별적 멤버를 위하여(예, seconds)

여러 가지 함수를 제공한다. 데이터베이스 서버에 있는 현재의 시스템 date / time은 NOW와 같은 함수를 사용하여 불러올 수 있다. The IBM Informix implementation에서는 sub-second precision을 필요로 하는 시스템을 위하여 시간의 정확성을 높일 수 있는 EXTEND와 FRACTION 함수를 제공하고 있다.

7.3.8 Data control

The Data Control Language (DCL)은 데이터에 접근하여 조작할 수 있는 이용자의 권한을 제공한다. 여기에는 두 가지의 주요 명령문이 있다:

1) GRANT; 사물의 운영에 대하여 1인 이상의 이용자에게 권한을 부여한다.
2) REVOKE; 디폴트 grant일 수도 있는 grant를 제거한다.

Example:

```
GRANT SELECT, UPDATE
 ON example
 TO some_user, another_user;

REVOKE SELECT, UPDATE
 ON example
 FROM some_user, another_user;
```

7.4 Procedural extensions

SQL은 관계형 데이터베이스에 들어있는 데이터를 쿼리하기 위한 특별한 목적용으로 디자인 되었다. SQL은 C나 BASIC과 같은 an imperative(명령) language가 아니라 a set-based, declarative(서술적) query language 이다. 그렇지만 Standard SQL의 확장을 통해 control-of-flow constructs와 같은 procedural programming language functionality을 추가하였다. 여기에는 다음과 같은 것들이 있다:

ANSI/ISO Standard	SQL/PSM	SQL/Persistent Stored Modules
Interbase / Firebird	PSQL	Procedural SQL
IBM DB2	SQL PL	SQL Procedural Language (implements SQL/PSM)
IBM Informix	SPL	Stored Procedural Language
IBM Netezza	NZPLSQL	(based on Postgres PL/pgSQL)
Microsoft / Sybase	T-SQL	Transact-SQL
Mimer SQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
MySQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
NuoDB	SSP	Starkey Stored Procedures
Oracle	PL/SQL	Procedural Language/SQL (based on Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL (based on Oracle PL/SQL)
PostgreSQL	PL/PSM	Procedural Language/Persistent Stored Modules (implements SQL/PSM)
Sybase	Watcom-SQL	SQL Anywhere Watcom-SQL Dialect
Teradata	SPL	Stored Procedural Language

the standard SQL/PSM extensions and proprietary SQL extensions에 따라서, procedural and object-oriented programmability을 다른 언어와 통합된 DMBS를 통해 많은 SQL platforms에서 이용할 수 있게 되었다. SQL 표준에서는 SQL/JRT extensions (SQL Routines and Types for the Java Programming Language)로 하여금 SQL 데이터베이스에 있는 Java code를 지원하도록 하고 있다. SQL server 2005는 the SQLCLR (SQL Server Common Language Runtime)를 사용하여 그 데이터베이스에서 관리하고 있는 .NET assemblies를 호스트하고 있다. 반면에, 이전의 버전의 SQL Server에서는 기본적으로 C로 작성된 unmanaged extended stored procedures만으로 한정하였다. PostgreSQL은 이용자로 하여금 Peril, Python, Tcl, and C와 같은 다양한 언어에서 함수를 사용할 수 있도록 하고 있다.

7.5 Criticism

SQL은 그것의 이론적 근거 - 관계형 모델과 그것의 tuple calculus – 에서 여러 가지 방법으로 벗어나 있다. 이 모델에서, 테이블은 tuples의 a set인 반면에, SQL에서 테이블과 쿼리의 결과는 rows의 리스트들이다: 동일한 로우가 여러 번 발생할 수 있고, 로우의 순서가 쿼리에서 결정될 수 있다(예, LIMIT 절에서). 이것이 공동의 실질적 관심사이면서 또한 논쟁의 주제이다. 더구나 (NULL과 views와 같은) 추가적 특징들이 관계형 모델에 직접적인 뿌리를 두지 않은 채로 소개됨으로써 그것들을 해석하는 것이 더욱 어렵게 되었다.

비평가들은 SQL이 엄격하게 본래의 기본을 나타내는 언어로 대체되어야 한다고 주장하고 있다. 예를 들어, The Third Manifesto를 보라. 다른 비평가들은 Datalog가 SQL에 비해 2개의 장점을 갖고 있다고 주장하고 있다: 프로그램의 이해와 유지를 용이하게 하는 보다 명확한 어의(cleaner semantics), 그리고 특히 반복적 쿼리(recursive queries)에서 더욱 표현적(more expressive)이라는 것이다

또 다른 비평은 SQL 이행이 벤더들 간에 호환되지 않으며 표준을 완벽하게 따를 필요도 없다는 것이다. 특정한 날짜와 시간의 구문에서, string concatenation(사슬, 연결), NULLs, and comparison case sensitivity는 상인마다 다르다. 특별한 예외가 표준을 지키려고 애쓰는 PostgreSQL 이다.

SQL에서 인기있는 이행은 공통적으로 DATE or TIME data types과 같은 Standard SQL의 기본적 특징의 지원을 생략하는 것이다. 가장 분명한 그러한 예들과 말하자면(incidentally) 가장 인기있는 상업적 그리고 전용 SQL DBMSs가 (DATE를 DATETIME처럼 사용하여 TIME type이 뺀) Oracles과 2008버전 이전의 MS SQL Server 이다. 결론적으로, SQL 코드는 데이터베이스 시스템 사이에서 거의 변형 없이 연결(port)되지 않는다.

데이터베이스 시스템 간에 이러한 호환성(portability)이 부족한 이유는 여러 가지가 있다:

1) The complexity and size of the SQL standard means that most implementors do not support the entire standard.
2) The standard does not specify database behavior in several important areas (e.g. indexes, file storage...), leaving implementations to decide how to behave.
3) The SQL standard precisely specifies the syntax that a conforming database system

must implement. However, the standard's specification of the semantics of language constructs is less well-defined, leading to ambiguity.

4) Many database vendors have large existing customer bases; where the newer version of the SQL standard conflicts with the prior behavior of the vendor's database, the vendor may be unwilling to break backward compatibility.

5) There is little commercial incentive for vendors to make it easier for users to change database suppliers (see vendor lock-in).

6) Users evaluating database software tend to place other factors such as performance higher in their priorities than standards conformance.

7.6 Standardization

SQL was adopted as a standard by the American National Standards Institute (ANSI) in 1986 as SQL-86 and the International Organization for Standardization (ISO) in 1987. Nowadays the standard is subject to continuous improvement by the Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee SC 32, Data management and interchange, which affiliate to ISO as well as IEC. It is commonly denoted by the pattern: ISO/IEC 9075-n:yyyy Part n: title, or, as a shortcut, ISO/IEC 9075.

ISO/IEC 9075 is complemented by ISO/IEC 13249: SQL Multimedia and Application Packages (SQL/MM), which defines SQL based interfaces and packages to widely spread applications like video, audio and spatial data.

Until 1996, the National Institute of Standards and Technology (NIST) data management standards program certified SQL DBMS compliance with the SQL standard. Vendors now self-certify the compliance of their products.[36]

The original standard declared that the official pronunciation for "SQL" was an initialism: /ˈɛs kjuː ˈɛl/ ("es queue el"). Regardless, many English-speaking database professionals (including Donald Chamberlin himself[37]) use the acronym-like pronunciation of /ˈsiːkwəl/ ("sequel"), mirroring the language's pre-release development name of "SEQUEL".

The SQL standard has gone through a number of revisions:

1989	SQL-89	FIPS127-1	Minor revision, in which the major addition were integrity constraints. Adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features (e.g. structured types). Support for embedding SQL in Java (SQL/OLB) and vice-versa (SQL/JRT).
2003	SQL:2003	SQL 2003	Introduced XML-related features (SQL/XML), *window functions*, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006	SQL 2006	ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it enables applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access

			ordinary SQL-data and XML documents.
2008	SQL:2008	SQL 2008	Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers. Adds the TRUNCATE statement.
2011	SQL:2011		

Interested parties may purchase SQL standards documents from ISO,[41] IEC or ANSI. A draft of SQL:2008 is freely available as a zip archive.[42]

The SQL standard is divided into nine parts.

1) ISO/IEC 9075-1:2011 Part 1: Framework (SQL/Framework). It provides logical concepts.
2) ISO/IEC 9075-2:2011 Part 2: Foundation (SQL/Foundation). It contains the most central elements of the language and consists of both mandatory and optional features.
3) ISO/IEC 9075-3:2008 Part 3: Call-Level Interface (SQL/CLI). It defines interfacing components (structures, procedures, variable bindings) that can be used to execute SQL statements from applications written in Ada, C respectively C++, COBOL, Fortran, MUMPS, Pascal or PL/I. (For Java see part 10.) SQL/CLI is defined in such a way that SQL statements and SQL/CLI procedure calls are treated as separate from the calling application's source code. Open Database Connectivity is a well-known superset of SQL/CLI. This part of the standard consists solely of mandatory features.
4) ISO/IEC 9075-4:2011 Part 4: Persistent Stored Modules (SQL/PSM) It standardizes procedural extensions for SQL, including flow of control, condition handling, statement condition signals and resignals, cursors and local variables, and assignment of expressions to variables and parameters. In addition, SQL/PSM formalizes declaration and maintenance of persistent database language routines (e.g., "stored procedures"). This part of the standard consists solely of optional features.
5) ISO/IEC 9075-9:2008 Part 9: Management of External Data (SQL/MED). It provides extensions to SQL that define foreign-data wrappers and datalink types to allow SQL to manage external data. External data is data that is accessible to, but not managed by, an SQL-based DBMS. This part of the standard consists solely of optional features.
6) ISO/IEC 9075-10:2008 Part 10: Object Language Bindings (SQL/OLB). It defines the syntax and semantics of SQLJ, which is SQL embedded in Java (see also part 3). The standard also describes mechanisms to ensure binary portability of SQLJ applications, and specifies various Java packages and their contained classes. This part of the standard consists solely of optional features, as opposed to SQL/OLB JDBC, which is not part of the SQL standard, which defines an API.[citation needed]
7) ISO/IEC 9075-11:2011 Part 11: Information and Definition Schemas (SQL/Schemata). It defines the Information Schema and Definition Schema, providing a common set of tools to make SQL databases and objects self-describing. These tools include the SQL object identifier, structure and integrity constraints, security and authorization specifications, features and packages of ISO/IEC 9075, support of features provided by

SQL-based DBMS implementations, SQL-based DBMS implementation information and sizing items, and the values supported by the DBMS implementations.[43] This part of the standard contains both mandatory and optional features.

8) ISO/IEC 9075-13:2008 Part 13: SQL Routines and Types Using the Java Programming Language (SQL/JRT). It specifies the ability to invoke static Java methods as routines from within SQL applications ('Java-in-the-database'). It also calls for the ability to use Java classes as SQL structured user-defined types. This part of the standard consists solely of optional features.
9) ISO/IEC 9075-14:2011 Part 14: XML-Related Specifications (SQL/XML). It specifies SQL-based extensions for using XML in conjunction with SQL. The XML data type is introduced, as well as several routines, functions, and XML-to-SQL data type mappings to support manipulation and storage of XML in an SQL database.[39] This part of the standard consists solely of optional features.[citation needed]

ISO/IEC 9075 is complemented by ISO/IEC 13249 SQL Multimedia and Application Packages. This closely related but separate standard is developed by the same committee. It defines interfaces and packages based on SQL. The aim is a unified access to typical database applications like text, pictures, data mining or spatial data.

1) ISO/IEC 13249-1:2007 Part 1: Framework
2) ISO/IEC 13249-2:2003 Part 2: Full-Text
3) ISO/IEC 13249-3:2011 Part 3: Spatial
4) ISO/IEC 13249-5:2003 Part 5: Still image
5) ISO/IEC 13249-6:2006 Part 6: Data mining
6) ISO/IEC 13249-8:xxxx Part 8: Metadata registries (MDR) (work in progress)

8. Database normalization

데이터베이스 정규화(Database normalization)는 과잉(redundancy)을 최소화하기 위하여 관계형 데이터베이스의 필드와 테이블을 조직하는 과정이다. 정규화에서는 대체로 커다란 테이블을 보다 작은 그리고 중복이 보다 적은 테이블로 나눈 다음에 그것들 간의 관련성을 정의한다. 이것의 목적은 한 필드의 추가, 삭제, 변경이 단지 한 테이블에서만 이루어진 다음에 정의된 관련성에 근거하여 그 데이터베이스의 나머지 테이블까지 파급되도록 데이터를 고립(isolate)시키는 것이다.

관계형 모델의 개발자인 Edgar F. Codd가 정규화의 개념을 소개하였으며, 그가 the First Normal Form (1NF) in 1970, the Second Normal Form (2NF) and Third Normal Form (3NF) in 1971를, 그리고 Codd and Raymond F. Boyce가 the Boyce-Codd Normal Form (BCNF) in 1974를 정의하였다. 비공식적으로, 관계형 데이터베이스 테이블은 그것이 제 3 정규형에 있으면, "정규화"되었다고 말하기도 한다. 대부분의 제 3 정규형의 테이블은 insertion, update, and deletion anomalies에서 자유롭다.

데이터베이스 디자인 가이드의 기준에서, 디자이너는 먼저 a fully normalized design을 제작한 다음에, 성능을 높이기 위해 선택적 denormalization을 실행하여야 한다고 말하고 있다.

8.1 Purpose

Edgar Frank "Ted" Codd in 1970가 정의한 the first normal form의 기본적인 목적은 first-order logic을 바탕으로 하는 "a "universal data sub-language"를 사용하여 데이터를 쿼리하고 조작하도록 하는 것이다. (SQL은 Codd가 심각한 결함이 있는 것으로 판단하고 있지만, 그러한 data sub-language의 한 예이다.

E.F. Codd가 "Further Normalization of the Data Base Relational Model"에서 언급했듯이, 1NF의 정규화의 목적은 다음과 같다:

1. 원하지 않는 삽입, 갱신, 삭제 의존성으로부터 한 집단의 관계(the collection of relations)를 자유롭게 하는 것;
2. 새로운 유형의 데이터를 소개함으로써 한 집단의 관계의 재구성에 대한 필요성을 절감시켜, 응용 프로그램의 life span을 늘리는 것.
3. 관계형 모델을 이용자에게 보다 정보적으로(informative) 만드는 것;
4. 한 집단의 관계들을 query statistics – 이 통계들은 시간이 지남에 따라 변화에 취약하다 – 에 중립적으로 만드는 것.

8.1.1 Free the database of modification anomalies

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
436	67 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

*An **update anomaly**. Employee 519 is shown as having different addresses on different records.*

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201
424	Dr. Newsome	29-Mar-2007	?

*An **insertion anomaly**. Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his details cannot be recorded.*

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

DELETE

*A **deletion anomaly**. All information about Dr. Giddens is lost if he temporarily ceases to be assigned to any courses.*

테이블을 변경(갱신, 삽입, 삭제)하려할 때, 원하지 않는 부수효과(side-effects)가 발생한다. 모든 테이블이 이러한 부수효과로 어려움을 겪는 것은 아니다; 그보다, 부수효과는 충분하게 정규화되지 않은 테이블에서만 발생한다. 불충분하게 정규화된 테이블은 다음과 같은 하나 이상의 특징을 가질 수 있다:

1) 똑같은 정보가 복수의 로우에 표현될 수 있다; 그러므로 그 테이블을 갱신하는 것은 논리적 불일치(logical inconsistencies)를 초래할 수 있다. 예를 들어, "Employees' Skills" table 에 있는 각 레코드는 an Employee ID, Employee Address, Skill을 갖고 있을 수 있다; 그러므로 특정한 종업원의 주소의 변경은 잠재적으로 복수의 레코드(one for each skill)에 적용될 필요가 있다. 만일 갱신이 충분하게 전달되지 않는다면 - 즉, 종업원의 주소가 어떤 레코드에서는 갱신되지만, 다른 것에서는 안 된다면, 그 테이블은 불일치 상태로 남게 된다. 특히, 이 같은 테이블은 이 종업원의 주소에 대한 질문에 답하는데 불란(conflicting)을 일으킨다. 이러한 현상을 **update anomaly**라 한다.

2) 어떤 사실이 결코 기록될 수 없는 환경이 존재한다. 예를 들어, "Faculty and Their

Courses" table의 한 레코드는 Faculty ID, Faculty Name, Faculty Hire Date, and Course Code를 갖고 있다. 그러므로 우리는 적어도 한 과목을 가르치는 교수에 대한 내역을 기록할 수 있으나, 아직 과목을 배정받지 못한 신임교수에 대한 내역을 기록할 수 없다. 왜냐하면, the Course Code가 null이기 때문이다. 이러한 현상을 **insertion anomaly**라 한다.

3) 어떤 환경에서, 어떤 사실을 표현하고 있는 데이터의 삭제는 전혀 다른 사실을 표현하고 있는 데이터의 삭제를 수반한다(necessitates). 위의 예인 "Faculty and Their Courses" table은 이러한 이상(anomaly)을 겪고 있다. 만일 교수에게 임시로 어떤 과목의 배정이 중지된다면, 우리는 효과적으로 그 교수를 삭제하기 위하여 그 교수 레코드의 지난 것들을 삭제해야만 한다. 이러한 현상을 **deletion anomaly**라 한다.

8.1.2 Minimize redesign when extending the database structure

새로운 유형의 데이터를 받아들이기 위하여 충분히 정규화된 데이터베이스 구조를 확대하고자 할 때, 기존에 존재했던 데이터베이스 구조는 거의 또는 전혀 변하지 않아야 한다. 결과적으로, 그 데이터베이스와 상호작용하는 어플들은 최소한으로 영향을 끼쳐야 한다. 이것은 table creation에 매우 유용하다.

8.1.3 Make the data model more informative to users

정규화된 테이블, 그리고 하나의 정규화된 테이블과 다른 것과의 관계는 실세계의 개념과 그것간의 상호연관성을 반영한다(mirror).

8.1.4 Avoid bias towards any particular pattern of querying

정규화된 테이블은 일반용도의 쿼링에 적합하다. 즉, 그 내역을 예상할 수 없는 미래의 쿼리를 포함하고 있는 테이블에 대하여 어떠한 쿼리도 지원한다는 의미이다. 대조적으로, 정규화되지 못한 테이블은 어떤 종류의 쿼리에는 적합(lend)하지만 다른 것에는 그렇지 않다.

예를 들어, 고객이 갖고 싶어하는 도서의 목록(wish-list)를 갖고 있는 온라인 서점을 생각해 보자. “이 고객이 무슨 책을 원하는가”와 같은 분명하고 예측된 쿼리를 위하여, 저자와 서명의 동일한 문자열(homogeneous string of authors and titles)로 된 테이블에 그 고객의 wishlist를 저장할 수 있다.

그렇지만, 이러한 디자인에서, 데이터베이스는 단지 하나의 단일 쿼리에 대해서만 대답할 수 있다. 스스로는 재미있지만 예측하지 못한 쿼리는 답할 수 없다: 가장 보고 싶어하는 책이 무엇인가? 어떤 고객이 WW II espinage에 관심을 갖고 있는가? 어떻게 Lord Byron이 동시대 시인과 필적하는가(stack up against)? 이러한 질문에 대한 답은 그 데이터베이스와 완전하게 독립된 특별한 adaptive tools로부터 얻어지는 것이 분명하다. 한 가지 도구는 그러한 쿼리를 특별하게 취급하도록 만든 소프트웨어일 수도 있다. 이러한 특별한 adaptive software는 단지 하나의 목적만을 가지고 있다: 결국 비-정규화 필드를 정규화로 만드는 것.

Unforeseen queries can be answered trivially, and entirely within the database framework, with a normalized table.

8.2 Example

고객의 신용카드 거래에 대한 다음과 같은 non-1NF representation과 같은 정규화되지 않는 데이터 구조에 들어있는 데이터를 쿼리하고 조작하는 것은 필요이상으로 복잡하다.

<table>
<tr><th>Customer</th><th>Transactions</th></tr>
<tr><td>Jones</td><td>
<table>
<tr><th>Tr. ID</th><th>Date</th><th>Amount</th></tr>
<tr><td>12890</td><td>14-Oct-2003</td><td>−87</td></tr>
<tr><td>12904</td><td>15-Oct-2003</td><td>−50</td></tr>
</table>
</td></tr>
<tr><td>Wilkinson</td><td>
<table>
<tr><th>Tr. ID</th><th>Date</th><th>Amount</th></tr>
<tr><td>12898</td><td>14-Oct-2003</td><td>−21</td></tr>
</table>
</td></tr>
<tr><td>Stevens</td><td>
<table>
<tr><th>Tr. ID</th><th>Date</th><th>Amount</th></tr>
<tr><td>12907</td><td>15-Oct-2003</td><td>−18</td></tr>
<tr><td>14920</td><td>20-Nov-2003</td><td>−70</td></tr>
<tr><td>15003</td><td>27-Nov-2003</td><td>−60</td></tr>
</table>
</td></tr>
</table>

반복적 그룹의 거래(a *repeating group* of transactions)가 각각의 고객에게서 나타나고 있다. 그러므로 고객의 거래에 관한 쿼리에 대한 자동 평가는 크게 다음과 같은 2 단계로 이루어지게 된다:

1. 한 그룹에 있는 개별적 거래들을 조사할 수 있도록 한명 이상의 고객들의 그러한 그룹을 해체한다(unpacking);
2. 1 단계의 결과를 근거호 쿼리 결과를 끌어낸다(deriving).

예를 들어, 2003년 10월에 이루어진 모든 고객의 모든 거래 총액을 알아보기 위하여, 그 시스템은 각 고객의 *Transactions* group을 먼저 해체한 다음에, 2003년 10월에 해당하는 거래의 *Date*에서 수집한 모든 거래의 *Amounts*를 계산하여야 한다.

Codd의 중요한 생각 중의 하나는 이러한 구조적 복잡성이 쿼리를 공식화하고(이용자와 어플에 의해) 그리고 평가하는(DBMS에 의해) 방식에 많은 power and flexibility를 갖게 함으로써 항상 완전하게 제거될 수 있다는 것이었다. 위의 구조를 정규화한 것은 다음과 같다:

Customer	Tr. ID	Date	Amount
Jones	12890	14-Oct-2003	−87
Jones	12904	15-Oct-2003	−50

Wilkins	12898	14-Oct-2003	-21
Stevens	12907	15-Oct-2003	-18
Stevens	14920	20-Nov-2003	-70
Stevens	15003	27-Nov-2003	-60

이제 각각의 로우는 개별적인 신용카드 거래를 나타내며, 그 DBMS는 10월에 해당하는 Date를 갖고 있는 모든 로우를 찾아서 그것들의 Amounts를 계산함으로써 간단하게 해답을 얻을 수 있다. 이 데이터의 구조는 DBMS에 직접적으로 각각의 값을 노출시킴으로써, 대등하게(on an equal footing) 모든 값에 자리를 부여하고 있기 때문에, 각각의 값은 잠재적으로 쿼리에 직접적으로 참여할 수 있다.; 반면에 이전의 상황에서는 어떤 값들은 특별하게 처리해야만 하는 하위 단계의 구조에 내재되어 있었다. 따라서 정규화 디자인은 스스로 일반적인 목적의 쿼리를 처리할 수 있는 반면에 비정규화(unnormalized) 디자인은 그렇게 하지 못한다.

8.3 Background to normalization: definitions

8.3.1 Functional dependency(함수적 의존성)

특정한 테이블에서, 만일 또는 단지 각각의 X 값이 정확하게 한 개의 Y 값하고만 결합한다면, 속성 Y는 속성 X의 집합(set)에 함수적으로 의존하고 있다고 말하며, (X -> Y)라고 표현한다. 예를 들어, 속성 “Employee ID”와 “Employee Date of Birth”을 포함하고 있는 “Employee” 테이블에서, 함수적 의존성 {Employee ID} -> {Employee Date of Birth}를 갖게 될 것이다.

8.3.2 Full functional dependency(완전한 함수적 의존성)

만일 그것이: a) 함수적으로 X에 의존하고 있고, 그리고 b) X의 어떤 독특한(proper) 하위 집합에 함수적으로 의존하지 않고 있다면, 그런 속성은 속성 X의 집합에 완전하게 함수적으로 의존하고 있다.

{Employee Address}는 {Employee ID, Skill}에 함수적으로 의존하고 있지만, 완전한 함수적 의존성을 갖고 있지는 않다. 그 이유는 이것은 {Employee ID}에도 의존하고 있기 때문이다. {Skill}을 제거한다 하더라도, 함수적 의존성은 아직까지 {Employee Address} 그리고 {Employee ID} 간에 유지되고 있다.

8.3.3 Transitive dependency(전의적 의존성)

전이적 의존성은 간접적인 함수적 의존성이며, X -> Z는 단지 X -> Y와 Y -> Z에 의해서만 이루어지는 의존성을 말한다.

8.3.4 Trivial(하찮은, 사소한, 일상적) functional dependency

일상적 의존성이란 그것의 superset에 대한 속성의 함수적 의존성을 말한다. {Employee Address} → {Employee Address}처럼, {Employee ID, Employee Address} → {Employee Address}는 일상적인 것이다.

8.3.5 Multivalued dependency(복수값 의존성)

복수값 의존성이란 테이블의 어떤 로우들이 있느냐에 따라 어떤 다른 로우들의 존재를 암시하는 제한요소(constraint)이다.

8.3.6 Join dependency(조인 의존성)

만일 T가 T의 속성인 하위 집합을 가지고 있는 각 테이블을 다수의 테이블과 결합하여 항상 다시 제작될 수 있다면, 테이블 T는 조인 의존성을 갖는다.

8.3.7 Superkey(슈퍼키)

슈퍼키란 속성들의 결합이며, 이것은 데이터베이스 레코드를 유일하게 식별하기 위하여 사용될 수 있다. 한 개의 테이블에는 다수의 슈퍼키가 있을 수 있다.

8.3.4 Candidate key(후보키)

후보키는 어떠한 외부(extraneous) 정보도 갖고 있지 않는 슈퍼키들 중에서 특별한 하위 집합이다: 이것은 최소한의 슈퍼키이다.

예를 들어, the fields <Name>, <Age>, <SSN> and <Phone Extension>으로 이루어진 테이블은 많은 잠재적 슈퍼키를 갖는다. 이것들 중에서 3가지 슈퍼키는 <SSN>, <Phone Extension, Name>그리고 <SSN, Name>이다. 이것들 중에서 단지 <SSN> 만이 후보키인데, 그 이유는 그 밖의 것들은 레코드를 유일하게 식별하는데 필요치 않은 정보를 포함하고 있기 때문이다. ('SSN'은 여기서 각각의 사람에게 부여된 유일한 사회보장번호(Social Security Number)를 말한다).

8.3.5 Non-prime attribute(비-으뜸 속성)

비-으뜸 속성이란 어떠한 후보키에서도 생기지 않는 속성이다. Employee Address는 "Employees' Skills" 테이블에 있는 비-으뜸 속성이다.

8.3.6 Prime attribute(으뜸 속성)

으뜸 속성은 역으로 어떤 후보키에서 생기는 속성이다.

8.3.7 Primary key

관계(relation)에 있는 하나의 후보키는 으뜸키로 설정될 수 있다. 이것이 일반적인 것(또는 어떤 상황에서 필요한 일)이지만, 엄격하게 기호적(notational)이어야 하며, 정규화와는 어떠한 관계도 갖지 않는다. 정규화와 관련해서, 모든 후보키들은 동일한 입장을 가지며 동일하게 처리된다.

8.4 Normal forms(정규형)

관계형 데이터베이스 이론인 NF란 논리적인 불일치성과 이상(logical inconsistencies and anomalies)에 대한 테이블의 면역성(immunity) 정도를 결정하는 표준을 제공하는 것이다. 테이블에 적용가능한 정규형이 높으면 높을수록, 그것의 취약성은 점점 더 줄어든다. 각 테이블은 한 개의 "highest normal form" (HNF)"를 갖는다: 정의하자면, 테이블은 그것의 HNF와 그것의 HNF 보다 낮은 모든 정규형의 요구조건을 항상 충족시켜야 한다 ; 또한 정

의에 의하면, 테이블은 그것의 HNF보다 더 높은 어떤 정규형의 요구조건을 충족시키지 못한다는 것이다.

정규형은 각 테이블에 적용될 수 있다; 데이터베이스 전체가 정규형 n이라고 말하는 것은 그것의 모든 테이블이 정규형 n이라고 말하는 것과 같다.

때때로 데이터베이스 디자인 초보자가 추측하길 정규화는 반복적으로 진행된다는 것이다, 다시 말해서, 1NF 디자인은 2NF를 위한 첫 번째 정규화이고, 그 다음으로 3NF 등으로 진행된다는 것이다. 이것은 전형적으로 정규화가 이루어지는 방법을 정확하게 설명한 것은 아니다. 어떤 예민하게 디자인된 테이블은 처음부터 3NF로 될 수 있다; 더구나 만일 그것이 3NF라면, 3NF인 HNF를 가질 확률이 높다. "보다 높은" 정규형태(3NF 이상)로의 도달은 대체로 추가적은 디자이너의 노력을 필요로 하진 않는데, 왜냐하면 3NF 테이블은 대체로 그것보다 높은 정규형의 요구조건을 충족시키기 위하여 어떠한 변경도 필요하기 않기 때문이다.

주요 정규형을 요약하면, 다음과 같다:

	Normal form	Brief definition
1 NF	First normal form	The domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain.
2 NF	Second normal form	No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key
3 NF	Third normal form	Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitive dependency is allowed.
EKNF	Elementary Key Normal Form	Every non-trivial functional dependency in the table is either the dependency of an elementary key attribute or a dependency on a superkey
BCNF	Boyce-Codd normal form	Every non-trivial functional dependency in the table is a dependency on a superkey
4 NF	Fourth normal form	Every non-trivial multivalued dependency in the table is a dependency on a superkey
5 NF	Fifth normal form	Every non-trivial join dependency in the table is implied by the superkeys of the table
DKNF	Domain/key normal form	Every constraint on the table is a logical consequence of the table's domain constraints and key constraints
6 NF	Sixth normal form	Table features no non-trivial join dependencies at all (with reference to generalized join operator)

8.4.1 First normal form

1NF는 관계형 데이터베이스에 있는 관계의 성질이다. 만일 각 속성의 도메인에 원자 값(atomic value)만이 들어있고, 각 속성의 값에 도메인에서 온 단일 값(single value)만을 갖고 있다면, 그 관계는 1NF에 있다.

Edgar Codd는 1971년 논문에서 1NF에 있는 관계를 정의하길, 그 관계의 도메인에 있는 어떠한 것도 그 자체의 집합들인 요소들을 갖지 않아야 한다고 하였다.

1NF는 관계형 데이터베이스에 있는 관계의 필수적인 성질이다. 그리고 데이터베이스 정규화란 첫 번째 정규화를 최소한의 요구조건으로 갖춘 표준 정규형태의 관계들로 데이터베이스를 표현하는 과정이다.

8.4.1.1 Examples

다음의 시나리오는 데이터베이스 디자이너가 어떻게 1NF를 위반할 수 있는지를 보여주고 있다.

a) Domains and values

디자이너가 고객의 이름과 전화번호를 레코드하기 바란다고 가정해 보자. 그는 다음과 같은 고객 테이블을 디자인한다:

customer

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659
789	Maria	Fernandez	555-808-9633

그런 다음에 디자이너는 몇몇 고객에게서 복수의 전화번호를 레코드할 필요가 있다는 것을 깨닫는다. 그는 이것을 처리하는 가장 간단한 방법으로, 다음과 같이 어떤 특정 레코드에 있는 “Telephone Number” 필드에 하나 이상의 값을 허용해야 한다고 생각한다:

customer

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659 555-776-4100
789	Maria	Fernandez	555-808-9633

그렇지만, 전화번호 칼럼이 12개 문자열의 도메인같이 어떤 전화번호-유형의 도메인에서 정의된다고 가정한다면, 위의 표현은 1NF가 아니며, 단지 아래와 같은 방식으로만 표현될 수 있다:

customer

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659
456	Jane	Wright	555-776-4100
789	Maria	Fernandez	555-808-9633

단일 필드에 복수의 값을 허용함으로써 이것은 1NF를 위반하고 있다. 전형적인 관계형 데이터베이스 관리 시스템에서는 한 테이블 안에 위와 같이 복수의 값을 포함하는 필드들을 허용하지 않는다.

8.4.1.2 A design that complies with 1NF: 1NF에 맞는 디자인

1NF에 분명하게 맞는 디자인은 다음과 같은 2개의 테이블을 사용하여 만드는 것이다: Customer Name 테이블과 Customer Telephone Number 테이블.

Customer Name

Customer ID	First Name	Surname
123	Robert	Ingram
456	Jane	Wright
789	Maria	Fernandez

Customer Telephone Number

Customer ID	Telephone Number
123	555-861-2025
456	555-403-1659
456	555-776-4100
789	555-808-9633

전화번호들의 반복이 이 디자인에서는 발생하지 않는다. 그 대신에, 각각의 Customer-to-Telephone Number 링크로 된 그 자체의 레코드가 발생한다. key인 고객 ID를 사용하면, 일-대-다의 관계가 두 테이블 간에 존재한다. "부모" 테이블인 고객 이름에 있는 레코드는 "자녀" 테이블인 고객 전화번호에 있는 많은 전화번호 레코드를 가질 수 있다. 그러나 각각의 전화번호는 단지 하나의 고객에게만 속한다. 따라서 주목해야 하는 것은 이 디자인을 통해 2 그리고 3NF의 추가적인 필요조건을 충족시킬 수 있다는 것이다.

8.4.1.3 Atomicity(원자성)

Edgar F. Codd의 1NF 정의는 'atomicity'의 개념을 참고하고 있다. "각각의 관계에서 정의하고 있는 도메인에 들어 있는 값들은 DBMS와 관련해서 원자적이야 한다."고 Codd는 말하였다. 또한 Codd는 하나의 원자 값을 (어떤 특별한 기능을 제외하고) "DBMS에 의해 더 이상 작은 조각으로 분해될 수 없는 것"으로 정의하고 있다. 이것은 하나의 필드는 그 안에 들어 있는 하나 이상의 유형으로 된 데이터의 부분(parts)으로 세분되지 않아야 한다는 의미인데, 왜냐하면 DBMS에서 한 부분이 나타내고 있는(의미하고 있는) 것을 동일한 필드에 있는 또 다른 부분들도 의존하고 있기 때문이다.

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.

H. Darwen과 C. Date가 제안하길, "원자 값"에 대한 Codd의 개념은 모호하며, 이러한 모호성은 1NF를 이해하는데 커다란 혼란만을 불러온다."고 하였다. 특히 "value that cannot be decomposed"이란 개념은 문제가 있는데, 왜냐하면 비록 있다하더라도 극소수만의 데이터 유형이 원자적이라는 의미로 받아들여질 수 있기 때문이다:

- A character string은 원자적이 아니라고 여겨질 수 있는데, 전형적으로 RDBMS에서는 그것을 하위 문자열로 분해할 수 있는 연산자를 제공하고 있기 때문이다.
- A fixed-point number 원자적이지 않다고 여겨질 수 있는데, 전형적으로 RDBMS에서는 그것을 정수와 분수로 분해할 수 있는 연산자를 제공하고 있기 때문이다.
- An ISBN은 원자적으지 않다고 여겨질 수 있는데, 그것에는 언어와 출판사 식별자가

포함되어 있기 때문이다.

Date가 제안하길, "the notion of atomicity *has no absolute meaning*": a value란 은 어떤 목적에 따라 원자로 여겨질 수도 있지만, 다른 목적을 위해서는 더 많은 기본적 요소의 집합으로도 고려될 수도 있다. 이러한 주장에 따르면, 1NF는 atomicity를 참고하여 정의할 수 없다. 어떤 있을 수 있는(conceivable) 데이터 유형(문자열 유형과 숫자 유형에서부터 arry type과 테이블 유형까지)의 칼럼들은 1NF 테이블에서 받아들일 수 있다 - 비록 항상 바람직한 것은 아니지만; 예를 들어, Customer Name 필드를 두 개의 독립된 필드인 First Name과 Surname 필드로 분리하는 것이 보다 바람하다.

1NF는, C. Date가 정의한 것처럼, relation-valued attributes(테이블 내의 테이블)을 허용하고 있다. Date가 주장하길, 한 테이블의 필드 속에 어떤 테이블을 포함하려는 수단으로 사용하는 relation-valued attributes은 매우 희귀한 경우에만 유용하다는 것이다.

8.4.1.4 1NF tables as representations of relations: 관계의 표현으로서 1NF 테이블

Date의 정의에 따라, 만일 테이블이 특히 그것은 다음과 같은 5가지의 조건을 만족시키는 "isomorphic(이종동형적) to some relation"이라면, 그것은 1NF에 있다:

1) 로우에 어떠한 top-to-bottom ordering도 존재하지 않는다.
2) 칼럼에 어떠한 left-to-right ordering도 존재하지 않는다.
3) 어떠한 중복된 로우도 존재하지 않는다.
4) 모든 row-and-column intersection에는 적용가능한 도메인(그 밖의 것에서는 절대로 아닌)으로부터 온 하나의 값만을 정확하게 갖는다.
5) 모든 칼럼은 규칙적(regular) 이다. [다시 말해서 로우는 로우 IDs, 사물 IDs, 또는 숨겨진 time-stamps(문서수발 일시기록)과 같은 숨겨진 구성요소를 절대로 갖지 않는다].

이러한 조건들의 어떤 것을 위반하는 것은 엄격하게 말해서 그 테이블이 관계형이 아니며 따라서 첫 번째 정규형이 아니라는 의미이다. 제 1 정규형의 이러한 정의를 충족시켜주지 못하는 테이블 또는 뷰의 예들은 다음과 같다:

1) 하나의 유일 키(unique)가 없는 테이블. 그러한 테이블은 조건 3의 위반으로 중복된 로우를 가질 수 있다.
2) 결과들을 특별한 순서로 나타내도록 강제하도록(mandates) 함으로써, row-ordering이 그 뷰의 고유적이고도 의미있는 모습이 되는 view. 이것은 조건 1의 위반이다. true relation에 있는 tuples는 서로 관련된 순서를 갖지 않는다.
3) 최소한 하나의 널이 들어갈 수 있는 속성(nullable attribute)을 가진 테이블. 널 가능 속성은 '모든 필드는 정확하게 그것의 칼럼 도메인에서 온 하나의 값만을 포함해야 한다' 는 조건 4의 위반이다. 그렇지만 주목해야 하는 것은 조건 4의 이 요인이 논쟁의 대상이라는 것이다. 왜냐하면, 이것은 널에 대한 분명한 조건을 제시한 Codd의 관계형 모델에 대한 후속 견해(later vision)에서 크게 이탈된 것이기 때문이다.

8.4.2 Second normal form

2NF는 데이터베이스 정규화에서 사용된 정규형이다. 2NF는 1971년에 E.F. Codd에 의해 처음 정의되었다.

1NF 테이블은 만일 그것이 2NF의 품격을 갖추기 위해서는 추가적인 기준을 만족시켜야만 한다. 좀 더 자세히 말해서, 만일 어떤 테이블이 1NF에 있고, 그 테이블의 어떠한 비-으뜸 속성도 그것의 후보 키들의 어떠한 하위 집합에도 의존하고 있지 않다면, 그 테이블은 2NF 이라는 것이다. 여기서 테이블의 비-으뜸 속성이란 그 테이블의 어떠한 후보 키의 일부분이 아닌 속성을 말한다.

요약하면, 어떤 테이블이 1NF에 있고 그것의 모든 비-으뜸 속성이 후보키의 전체에 의존하고 있다면, 그것은 2NF이다.

8.4.2.1 Example

employees' skills을 나타내고 있는 다음의 테이블을 보자:

Employees' Skill

Employee	Skill	Current Work Location
Brown	Light Cleaning	73 Industrial Way
Brown	Typing	73 Industrial Way
Harrison	Light Cleaning	73 Industrial Way
Jones	Shorthand	114 Main Street
Jones	Typing	114 Main Street
Jones	Whittling	114 Main Street

{Employee} 나 {Skill} 둘 다 이 테이블의 후보키가 아니다. 왜냐하면, 특정한 종업원이 한번이상(그는 복수의 기술을 가질 수 있다) 나타날 뿐만 아니라, 특정한 기술 역시 한번이상(다수의 종업원이 그 기술을 가질 수 있다) 나타나기 때문이다. 그렇지만, 단지 composite key {Employee, Skill}는 이 테이블의 후보키로서 자격을 갖추고 있다.

나머지 속성인 Current Work Location은 단지 후보키의 일부분인 즉 Employee에만 의존하고 있다. 그러므로 이 테이블은 2 NF가 아니다. Current Work Locations에 나타나 있는 중복성(redundancy)을 주목하라: 여기서 Jones가 114 Main Street에서 일한다고 3번 표현했으며, Brown은 73 Industrial Way에서 근무한다고 2번 표현했다. 이 같은 중복성으로 인하여 그 테이블을 갱신이상(update anomalies)에 취약하게 된다: 예를 들어, "**Shorthand**"와 "**Typing**" 레코드에서 Jone의 work location을 갱신할 수 있으나, 그의 "**Whittling(나무깍기)**" 레코드는 갱신 못 할 수 있다. 그 결과 데이터는 다음과 같은 질문에 대하여 모순된 답을 제공하게 된다: "Jone의 current work location은 무엇입니까?"

이런 디자인에 대한 2NF 대안은 동일한 정보를 두 개의 테이블로 다음과 같이 표현하는 것이다: 후보키 {Employee}를 갖는 Employees 테이블과 후보키 {Employee, Skill}를 갖는 Employees's Skills 테이블.

Employees

Employee	Current Work Location
Brown	73 Industrial Way
Harrison	73 Industrial Way
Jones	114 Main Street

Employees' Skills

Employee	Skill
Brown	Light Cleaning
Brown	Typing
Harrison	Light Cleaning
Jones	Shorthand
Jones	Typing
Jones	Whittling

이제 이 테이블들은 어떠한 것도 갱신이상으로 어려움을 겪지 않는다.

그렇지만 모든 2NF 테이블들이 갱신이상에서 자유로운 것은 아니다. 갱신이상으로 어려움을 겪는 2NF의 예는 다음과 같다:

Tournament Winners

Tournament	Year	Winner	Winner Date of Birth
Des Moines Masters	1998	Chip Masterson	14 March 1977
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

비록 Winner와 Winner Date of Birth가 key {Tournament, Year}의 일부에 의해서가 아니라 전체에 의해 결정되어지더라도, 특별한 Winner / Winner Date of Birth의 결합은 다수의 레코드에서 중복적으로 나타난다. 이것은 갱신이상을 발생시킨다; 만일 갱신이 일관적으로 수행되지 않는다면, 어떤 특별한 승리자는 두 개의 서로 다른 출생날짜를 갖는 것으로 나타날 것이다.

이것의 근본적인 문제는 Winner Date of Birth 속성이 의존하고 있는 전이적 의존성 (transitive dependency) 때문이다. Winner Date of Birth는 실재로 Winner에 의존하며, 그 다음에 키 Tournment / Year에 의존한다.

이 문제는 3NF에서 다룬다.

8.4.2.2 2NF and candidate keys: 2NF와 후보키

으뜸키에 대하여 어떠한 부분적 함수적 의존성을 갖고 있지 않은 테이블은 전형적으로 2NF이지만 꼭 그런 것만은 아니다. 으뜸키 이외에도, 테이블은 다른 후보키들을 가질 수 있다: 분명히 말해서, 어떠한 비-으뜸 속성도 이러한 후보키의 어떤 것에 대하여 part-key dependencies을 갖지 않아야 한다.

다음의 테이블에는 복수의 후보키가 나타나 있다:

Electric Toothbrush Models

Manufacturer	Model	Model Full Name	Manufacturer Country
Forte	X-Prime	Forte X-Prime	Italy
Forte	Ultraclean	Forte Ultraclean	Italy
Dent-o-Fresh	EZbrush	Dent-o-Fresh EZbrush	USA
Kobayashi	ST-60	Kobayashi ST-60	Japan
Hoch	Toothmaster	Hoch Toothmaster	Germany
Hoch	X-Prime	Hoch X-Prime	Germany

비록 디자이너가 {Model Full Name}을 으뜸키로 정한다 하더라도, 이 테이블은 2NF에 있지 않다. {Manufacturer, Model} 또한 후보키이며, Manufacturer Country는 그것의 올바른 하위집합에 의존하고 있다: Manufacturer. 이 디자인을 2NF로 변경하기 위해서는 다음과 같은 2 개의 테이블이 필요하다:

Electric Toothbrush Manufacturers

Manufacturer	Manufacturer Country
Forte	Italy
Dent-o-Fresh	USA
Kobayashi	Japan
Hoch	Germany

Electric Toothbrush Models

Manufacturer	Model	Model Full Name
Forte	X-Prime	Forte X-Prime
Forte	Ultraclean	Forte Ultraclean
Dent-o-Fresh	EZbrush	Dent-o-Fresh EZbrush
Kobayashi	ST-60	Kobayashi ST-60
Hoch	Toothmaster	Hoch Toothmaster
Hoch	X-Prime	Hoch X-Prime

8.4.3 Third normal form

3NF란 (a) 엔티티가 2NF에 있고, (b) 테이블의 모든 속성은 단지 으뜸키에만 확실하게 의존한다는 참조무결성(Referential integrity)을 지키도록 하여, 데이터의 이중성(duplication)을 줄이도록 데이터베이스를 정규화하는 3번째 디자인 과정이다.

8.4.3.1 Definition of third normal form

3NF는 데이터베이스 정규화에서 사용되는 정규형이다. 3NF는 1971년 E.F. Codd에 의해 처음 정의되었다. Codd의 정의에서 언급한 것은 어떤 테이블이 만일 다음과 같은 조건을 유지한다면 그것은 3NF에 있다는 것이다:

1) 관계 R(테이블)이 2NF에 있다.
2) R의 모든 비-으뜸 속성은 R의 모든 superkey에 비-전이적(non-transitively) 의존성을 갖고 있다.

R의 비-으뜸 속성은 R의 어떠한 후보키에도 속하지(belong) 않는 속성이다. 전이적 의존성이란 실재적으로는 X → Y 그리고 Y → Z (그렇지만, Y → X인 경우가 아니다)이지만, 간접적으로 X → Z (X가 Z를 결정한다)인 함수적 의존성을 말한다.

3NF는 Codd의 정의와는 같지만 1982년 C. Zaniolo에 의해 다르게 표현되었다. 이 정의에서 말하는 것은 테이블이 단지 그것의 함수적 의존성 X → A의 각각에 대하여 적어도 다음과 같은 조건 중 하나를 유지한다면 그것은 3NF에 있다는 것이다.

1) X는 A를 포함한다(즉, X → A는 사소한(trivial) 함수적 의존성이다).
2) X가 슈퍼키다.
3) A와 X간의 set difference(차집합)를 나타내는 A-X의 모든 요소는 으뜸 속성이다(다시 말해서, A-X에 있는 각 속성은 어떤 후보키 속에 포함된다).

Zaniolo의 정의는 3NF와 더욱 엄격한 BCNF(Boyce-Codd normal form) 간의 차이에 대하여 분명히 밝히고 있다. BCNF에서는 세 번째 대안(A와 X 간의 set difference인 "A-X의 모든 요소는 으뜸 속성이다")을 간단하게 제거하는 것이다.

8.4.3.2 "Nothing but the key"

법정에서 참된 증언을 하는 전통적 서약과 마찬가지로, 3NF에 대한 Codd의 정의에서 기억할만한 말은 B. Kent에 의해 제공되었다: "[모든] non-key[속성]는 the key, whole key, 그리고 단지(nothing but) the key에 대한 fact만을 제공해야만 한다." 한 가지 common variation이 다음과 같은 맹세와 더불어 이런 정의를 보완하고 있다: "so help me Codd".

"the key"의 존재를 밝히는 것이 그 테이블이 1NF에 있다는 것을 확인하는 것이며; 비-키 속성들이 "the whole key"에 의존함을 밝히는 것이 2NF이라는 것을 확인하는 것이고; 추가로 비-키 속성이 "nothing but the key"에 의존함을 밝히는 것이 3NF에 있다는 것을 확인하는 것이다.

C. Date는 Kent의 요약에 대해 3NF의 "an intuitively attractive characterization"으로 말하고 있으며, 그것을 조금난 바꾸면 좀 더 강력한 Boyce-Codd 정규형의 정의 - "각각의 속성은 the key, the whole key, and nothing but the key에 대한 사실을 표현해야만 한다."에 도움을 줄 수 있다고 하였다. 3NF는 단지 비-키 속성이 키들에 의존한다는 것을 확인하는 데만 관심을 갖기 때문에, 이러한 3NF의 정의는 Date의 BCNF의 variation보다 약하다. (키들이거나 키의 부분들)인 으뜸 속성들은 함수적으로 결코 의존적이지 않아야 한다: 그것들 각각은 스스로가 그 키의 일부이나 전체라는 뜻으로 그 키에 대한 사실을 표현하고 있다. (여기서 주목해야 하는 것은 어떤 그 같은 키의 각 부분이 "whole key" 절(clause)을 위반하게 되므로, 모든 속성에 이것을 적용하는 것은 은연중에 복합후보키를 금지하게 될 것이기 때문에, 이 규칙은 함수적으로 의존하는 속성들에게만 적용된다는 것이다.

3NF의 조건을 충족시키지 못하는 2NF의 예는 다음과 같다:

Tournament Winners

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

이 테이블의 각 로우는 특별한 Year에 특별한 Tournament의 우승자를 알려져야하기 때문에, 복합키 {Tournament, Year}는 어떤 로우를 유일하게 식별하기 위하여 필요한 최소한의 속성 집합이다. 즉, {Tournament, Year}는 이 테이블의 후보키이다.

3NF의 위반이 발생했는데, 왜냐하면 비-으뜸 속성인 Winner Date of Birth 가 비-으뜸 속성인 Winner를 거쳐서 후보키 {Tournament, Year}에 전이적으로 의존하고 있기 때문이다. Winner Date of Birth가 Winner에 함수적으로 의존하고 있다는 사실은 다른 레코드에 다른 생년월일을 갖고 동일한 사람이 등장하는 것을 막을 조치가 아무 것도 없으므로, 그 테이블을 logical inconsistencies(논리적 모순)으로 가도록 취약하게 만든다.

3NF 위반없이 동일한 사실을 표현하기 위하여 이 테이블은 다음과 같이 두 개로 나누어야 한다:

Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Des Moines Masters	1999	Al Fredrickson
Indiana Invitational	1999	Chip Masterson

Winner	Date of Birth
Chip Masterson	14 March 1977
Al Fredrickson	21 July 1975
Bob Albertson	28 September 1968

이제 3NF에 있는 두 개 모두의 테이블에서 갱신이상이 일어날 수 없다.

8.4.3.3 Derivation of Zaniolo's conditions

위에서처럼, 1982년에 Carlo Zaniolo가 제시한 3NF의 정의는 다음과 같은 방식으로 입증되었다: Let X → A be a nontrivial FD (즉, X가 A를 포함하지 않는 것) and let A be a non-key attribute. Also let Y be a key of R. Then Y → X.

8.4.3.4 Normalization beyond 3NF

대부분의 3NF 테이블들은 update, insertion, and deletion anomalies에서 자유롭다. 3NF 테이블의 어떤 유형들은 현실에서는 거의 만나기 힘들지만 이러한 이상들에 영향을 받는다; 이러한 테이블들은 Boyce-Codd normal form (BCNF)의 결함을 가지고 있거나 비록

BCNF를 충족시키더라도 4NF나 5NF와 같은 보다 고차원적인 정규형에는 결함을 갖고 있다.

8.4.4 EKNF(Elementary Key Normal Form)

Elementary Key Normal Form (EKNF)은 3NF보다 미묘하게 향상된 형태이다. 정의하자면, EKNF 테이블들 역시 3NF에 있지만, 하나 이상의 유일한 복합키가 있고, 이것들이 서로 중복될 때 이러한 형태가 발생한다. 이러한 경우는 중복된 칼럼들에 잉여정보의 발생 원인이 될 수 있다.

만일 그리고 단지 모든 기본적인 함수적 의존성이 whole keys에서 시작되거나 또는 기본 키 속성(elementary key attributes)에서 끝난다면, 그 테이블은 EKNF에 있다.

X->Y 형태인 모든 완전한 nontrivial 함수적 의존성에서, X는 키이거나 또는 Y는 기본 키이거나 그것의 일부이다.

8.4.4.1 Example

최상의 정규형이 EKNF인 테이블의 예는 아래의 Boyce-Codd normal form#Achievability of BCNF를 참조하기 바란다.

8.4.5 BCNF(Boyce-Codd Normal Form)

Boyce-Codd normal form (or BCNF or 3.5NF)은 데이터베이스 정규화에 사용되는 정규형이다. 이것은 3NF보다는 조금 더 강력한 것이다. BCNF는 Raymond F. Boyce and Edgar F. Codd에 의해 1974년에 개발되었으며, 원래 정의한 것처럼 3NF까지 다루지 못했던 어떤 유형의 이상(anomaly)에 초점을 맞추고 있다.

만일 관계형 스키마가 BCNF에 있다면, 비록 다른 유형의 과잉이 현재 존재하더라도 함수적 의존성에 의존하고 있는 모든 과잉은 제거된다. 만일 그리고 단지 그것의 의존성 X->Y의 모든 것과 관련해서 적어도 다음과 같은 조건들 중에서 하나을 유지하고 있다면, 그 관계형 스키마 R은 BCNF에 있다:

1) X → Y is a trivial functional dependency (Y ⊆ X)
2) X is a superkey for schema R

8.4.5.1 3NF tables not meeting BCNF (Boyce-Codd normal form)

3NF 테이블이 BCNF의 요구조건을 만족시키지 않는 경우는 매우 드물다. 다수의 중복된 후보키를 갖고 있지 않는 3NF 테이블은 BCNF에 확실하게 있는 것이다. 그것의 함수적 의존성이 무엇인가에 따라서, 2개 이상의 중복된 후보키를 갖고 있는 3NF 데이블은 BCNF에 있을 수도 있고 그렇지 않을 수도 있다. BCNF를 충족시키지 못하는 3NF의 예는 다음과 같다:

Today's Court Bookings

Court	Start Time	End Time	Rate Type
1	09:30	10:30	SAVER
1	11:00	12:00	SAVER

1	14:00	15:30	STANDARD
2	10:00	11:30	PREMIUM-B
2	11:30	13:30	PREMIUM-B
2	15:00	16:30	PREMIUM-A

- 테이블의 각 로우는 한 개의 하드코드(코트 1)와 한 개의 잔디코드(코트 2)를 갖고 있는 테니스 클럽의 코트 예약(court booking)을 나타낸다.
- 예약은 코트별 그리고 코트예약시간별로 이루어진다.
- 추가로, 각각의 예약은 그것과 관련된 Rate Type을 갖는다. 이것에는 4가지의 분명한 rate types가 있다:
  - SAVER; 회원이 코트 1을 예약.
  - STANDARD; 비회원이 코트 1을 예약.
  - PREMIUM-A; 회원이 코트 2를 예약.
  - PREMIUM-B; 비회원이 코트 2를 예약.

이 테이블의 수퍼키는 다음과 같다:

- S_1 = {Court, Start Time}
- S_2 = {Court, End Time}
- S_3 = {Rate Type, Start Time}
- S_4 = {Rate Type, End Time}
- S_5 = {Court, Start Time, End Time}
- S_6 = {Rate Type, Start Time, End Time}
- S_7 = {Court, Rate Type, Start Time}
- S_8 = {Court, Rate Type, End Time}
- S_T = {Court, Rate Type, Start Time, End Time}, the trivial superkey

위의 테이블에서 비록 Start Time과 End Time 속성들이 각각 중복된 값을 갖지 않는다 하더라도, 우리는 어떤 다른 날짜에 코트 1과 코트 2에 대한 두 가지 서로 다른 예약이 동시에 시작되거나 동시에 끝난다는 것을 시인해야만 한다는 것에 주목하여야 한다. 이것이 왜 {Start Time}과 {End Time}이 이 테이블의 수퍼키로 고려될 수 없는지에 대한 이유이다.

그렇지만, 단지 S_1, S_2, S_3 and S_4 는 후보키들인데(즉, 이 관계에 있어서 최소한의 수퍼키들), 왜냐하면 예를 들어 $S_1 \subset S_5$, so S_5 는 후보키가 될 수 없기 때문이다. 2NF에서 후보키에 대한 비-으뜸 속성(다시 말해서, 어떠한 후보키에서도 발생하지 않는 속성)의 부분적 함수적 의존성을 금지하고 있다는 것과 3NF에서 후보키에 대한 비-으뜸 속성의 전이적 함수적 의존성을 금지하고 있다는 것을 상기하라.

Today's Court Bookings 테이블에서, 어떠한 비-으뜸 속성들고 존재하지 않는다: 즉, 모든 속성들은 어떤 후보키에 속하고 있다. 그러므로 그 테이블은 2NF와 3NF 둘 다에 해당된다. 그렇지만, 이 테이블은 BCNF에 해당되지는 않는데, 그 이유는 결정 속성(the determining attribute (Rate Type))이 후보키도 그리고 후보키의 superset도 아닌, Rate Type → Court 의존성 때문이다. Rate Type → Court 의존성은 Rate Type가 단지 단일 Court에만 적용되어야할 때만 이루어진다(respected).

다음의 디자인은 BCNF를 충족시키기 위하여 수정될 수 있다:

Rate Types

Rate Type	Court	Member Flag
SAVER	1	Yes
STANDARD	1	No
PREMIUM-A	2	Yes
PREMIUM-B	2	No

Today's Bookings

Rate Type	Start Time	End Time
SAVER	09:30	10:30
SAVER	11:00	12:00
STANDARD	14:00	15:30
PREMIUM-B	10:00	11:30
PREMIUM-B	11:30	13:30
PREMIUM-A	15:00	16:30

Rate Types 테이블용 후보키들은 {Rate Type} 과 {Court, Member Flag} 이다: Today's Bookings 테이블의 후보키들은 {Rate Type, Start Time} 그리고 {Rate Type, End Time} 이다. 이들 둘 다 BCNF에 있다. {Rate Type} 이 Rate Types 테이블의 키일 때, 두 개의 서로 다른 코트와 연결된 Rate Type을 갖는 것은 불가능하다. 그러므로 Rate Types 테이블에 있는 키로서 {Rate Type}을 사용함으로써, 최초의 테이블에 영향을 끼치는 이상(anomaly)을 제거할 수 있다.

8.4.5.2 Achievability(완수성) of BCNF

어떤 경우에, 비-BCNF 테이블은 BCNF를 만족시켜서 최초의 테이블에서 유지되었던 의존성을 보존하고 있는 테이블들로 분해될 수 없다. 1979년에 Beeri 와 Bernstein은 예를 들어, 한 세트의 함수적 의존성 {AB → C, C → B}는 BCNF 스키마에 의해 표현될 수 없다는 것을 보여주었다. 따라서 처음의 3가지의 정규형과 달리, BCNF는 항상 완성될 수 없다.

함수적 의존성이 {AB → C, C → B} pattern을 따르는 다음의 비-BCNF 테이블을 생각해 보자:

Nearest Shops

Person	Shop Type	Nearest Shop
Davidson	Optician	Eagle Eye
Davidson	Hairdresser	Snippets
Wright	Bookshop	Merlin Books
Fuller	Bakery	Doughy's
Fuller	Hairdresser	Sweeney Todd's
Fuller	Optician	Eagle Eye

각각의 Person / Shop Type 결합과 관련해서, 이 테이블이 우리에게 말하고 있는 것은 이런 종류의 상점은 지리적으로 사람들의 집과 가장 가까이 있다는 것이다. 우리는 하나의 상점이 한 개 이상의 유형일 수 없다고 간단하게 가정하자.

이 테이블의 후보키들은 다음과 같다:

- {Person, Shop Type}
- {Person, Nearest Shop}

모두 3개의 속성이 으뜸 속성이기 때문에(즉, 후보키에 속한다), 이 테이블을 3NF에 있다. 그렇지만 이 테이블이 BCNF에는 속하지 않는데 Shop Type 속성이 함수적으로 비-수퍼키인 Nearest Shop에 의존하고 있기 때문이다.

BCNF의 위반이라고 하는 것은 이 테이블이 이상(anomalies)에 영향을 받기 때문이다. 예를 들어, Eagle Eye는 “Davidson” 레코드에서 Shop Type “Optician”을 사용하고 있는 동안, 그것의 “Fuller” 레코드에서 “Optometrist”로 바뀐 Shop Type을 가질 수도 있다. 이것은 다음과 같은 질문에 모순된 해답을 제공할 것이다: “무엇이 Eagle Eye's Shop Type인가?”. 각 상점이 일단 한 번 선호하는 Shop Type을 가짐으로써 발생 가능한 이상(anomalies)들을 예방할 수 있다:

Shop Near Person

Person	Shop
Davidson	Eagle Eye
Davidson	Snippets
Wright	Merlin Books
Fuller	Doughy's
Fuller	Sweeney Todd's
Fuller	Eagle Eye

Shop

Shop	Shop Type
Eagle Eye	Optician
Snippets	Hairdresser
Merlin Books	Bookshop
Doughy's	Bakery
Sweeney Todd's	Hairdresser

이렇게 수정된 디자인에서, "Shop Near Person" 테이블은 {Person, Shop}라는 후보키를 갖으며, "Shop" 테이블은 {Shop}이란 후보키를 갖는다. 불행하게도, 비록 이 디자인이 BCNF에 적합하다 하더라도 다양한 토대(grounds)를 받아들일 순 없다: 이것은 동일한 사람에 맞서는(against) 동일한 유형의 복수의 상점을 우리로 하여금 레코드하게 한다. 바꿔 말해서, 그것의 후보키들은 함수적 의존성 {Person, Shop Type} → {Shop}가 올바르다는 것(respected)을 보증하지는 않는다.

이러한 모든 이상을 제거하는(그러나 BCNF에 따르지 않는) 디자인이 가능하다. 이런 디자인은 Elementary Key Normal Form으로 알려진 새로운 정규형이다. 이 디자인은 위에서 설

명한 Shop 테이블에 의해 보완된 최초의 Nearest Shops 테이블로 구성된다. Bernstein's schema generation algorithm에 의해 만들어진 이 테이블 구조는 비록 그 같은 3NF의 개량형이 이 알고리즘이 디자인될 당시에는 인정받지 못했더라도 사실상 EKNF이다.

Nearest Shops

Person	Shop Type	Nearest Shop
Davidson	Optician	Eagle Eye
Davidson	Hairdresser	Snippets
Wright	Bookshop	Merlin Books
Fuller	Bakery	Doughy's
Fuller	Hairdresser	Sweeney Todd's
Fuller	Optician	Eagle Eye

Shop

Shop	Shop Type
Eagle Eye	Optician
Snippets	Hairdresser
Merlin Books	Bookshop
Doughy's	Bakery
Sweeney Todd's	Hairdresser

만일 참조무결성 조건(referential integrity constraint)이 첫 번째 테이블에서 온 {Shop Type, Nearest Shop}이 두 번째 테이블에서 온 {Shop Type, Shop}을 참조해야만 한다는 효과(effect)를 정의한 것이라면, 앞에서 기술한 데이터 이상을 예방할 수 있다.

8.4.6 Fourth normal form

4NF는 데이터베이스 정규화에 사용되는 정규형이다. 1977년 Ronald Fagin이 소개한 4NF는 BCNF의 다음 단계인 정규화이다. 2, 3, 그리고 Boyce-Codd 정규형이 함수적 의존성과 관련이 있는 반면에, 4NF는 복수값 의존성(multivalued dependency)이라고 하는 보다 일반적인 유형의 의존성과 관련이 있다. 만일 그리고 단지 그것의 non-trivial multivalued dependencies X ->> Y의 모든 것에 대하여, X가 수퍼키 - 즉, X가 후보키이거나 그것의 superset라면, 그 테이블은 4NF에 있다.

8.4.6.1 Multivalued dependencies

관계형 데이터베이스 테이블의 column headings를 3개의 disjoint(해체) groupings X, Y, Z로 나눌 수 있다면, 어떤 특별한 로우의 입장(context)에서 우리는 헤딩의 각 그룹 아래에 있는 데이터를 각각 x, y, z로 부를 수 있다. multivalued dependency X ->>(two head right arrow) Y는 만일 우리가 테이블에서 실재로 발생하는 어떤 x를 선택하여(이러한 선택을 xc라고 부르자), 그 테이블에서 발생하는 모든 xcyz 결합의 리스트를 작성(compile)한다면, xc는 z와 상관없이 똑같은 y entries와 결합하는 것을 발견할 것이라는 의미이다. 따라서 본질적으로 z란 존재는 y의 잠재적 값을 제한할 수 있는 어떠한 쓸모있는 정보도 제공하지 않는다.

A trivial multivalued dependency X ->> Y는 Y가 X의 subset이거나 X와 Y가 함께

그 관계의 모든 속성 세트를 형성할 때 존재한다.

함수적 의존성은 multivalued dependency의 특별한 경우이다. 함수적 의존성 X → Y에서, 모든 x는 정확하게 하나의 y를 결정하며, 결코 하나이상은 아니어야 한다.

8.4.6.2 Example

Pizza Delivery Permutations

Restaurant	Pizza Variety	Delivery Area
A1 Pizza	Thick Crust	Springfield
A1 Pizza	Thick Crust	Shelbyville
A1 Pizza	Thick Crust	Capital City
A1 Pizza	Stuffed Crust	Springfield
A1 Pizza	Stuffed Crust	Shelbyville
A1 Pizza	Stuffed Crust	Capital City
Elite Pizza	Thin Crust	Capital City
Elite Pizza	Stuffed Crust	Capital City
Vincenzo's Pizza	Thick Crust	Springfield
Vincenzo's Pizza	Thick Crust	Shelbyville
Vincenzo's Pizza	Thin Crust	Springfield
Vincenzo's Pizza	Thin Crust	Shelbyville

각 로우가 나타내는 것은 한 특정한 식당에 특정한 지역에 특정하고 다양한 피자를 배달할 수 있다는 것이다.

이 테이블은 어떠한 비-키 속성도 갖고 있지 않은데, 그 이유는 단지 키가 {Restaurant, Pizza Variety, Delivery Area}이기 때문이다. 그러므로 이것은 BCNF까지의 모든 정규형을 만족시키고 있다. 만일 우리가 가정하길, 식당에서 제공하는 피자의 종류는 배달지역에 영향을 받지 않는다면(즉, 식당이 모든 배달지역에 모든 다양한 피자를 배달한다), 이것은 4NF를 충족시키지 못한다. 문제는 {Restaurant} 속성(수퍼키가 아닌)과 관련해서 두 개의 non-trivial multivalued dependencies이 나타난다는 것이다. 이런 의존성은 다음과 같다:

- {Restaurant} ↠ {Pizza Variety}
- {Restaurant} ↠ {Delivery Area}

non-superkey에 대한 이러한 non-trivial multivalued dependencies은 식당에서 제공하는 피자의 종류는 식당에서 배달할 수 있는 지역과는 독립적이라는 사실에 영향을 끼친다. 즉, 이 경우에는 그 테이블에 잉여성이 발생한다: 예를 들어, 우리는 3번 말해야 한다; A1 피자는 Stuffed Crust를 제공하며, 만일 A1 Pizza가 Cheese Crust 피자를 만들기 시작한다면 우리는 A1 Pizza의 배달지역의 각각에 복수의 로우를 추가할 필요가 있을 것이다. 더구나 부정확하게 이것을 할 수 있는 예방조치가 우리에게는 아무 것도 없다: 우리는 A1 Pizza의 배달지역 모두가 아니라 한 곳만을 위하여 Cheese Crust 로우들을 추가할 수도 있다. 따라서 the multivalued dependency {Restaurant} ↠ {Pizza Variety}를 지키지 못하게 된다.

이러한 이상의 가능성을 제거하기 위하여, 우리는 배달지역에 대한 사실들로부터 어떤 다른 테이블 속으로 제공되는 종류(varieties)에 대한 사실을 마련해야만 하며, 이런 경우에 2 테

이블 모두 4NF의 조건을 만족시켜야 한다.

Varieties By Restaurant

Restaurant	Pizza Variety
A1 Pizza	Thick Crust
A1 Pizza	Stuffed Crust
Elite Pizza	Thin Crust
Elite Pizza	Stuffed Crust
Vincenzo's Pizza	Thick Crust
Vincenzo's Pizza	Thin Crust

Delivery Areas By Restaurant

Restaurant	Delivery Area
A1 Pizza	Springfield
A1 Pizza	Shelbyville
A1 Pizza	Capital City
Elite Pizza	Capital City
Vincenzo's Pizza	Springfield
Vincenzo's Pizza	Shelbyville

대조적으로, 만일 한 식당에서 제공되는 피자의 종류가 때때로 하나의 배달지역에서부터 다른 지역으로 적법하게 변한다면, 최초의 3-칼럼 테이블은 4NF를 만족시킬 것이다.

8.4.6.3 4NF in practice

Margaret S. Wu의 1992년 논문에서 전형적으로 데이터베이스 정규화의 강의에서 4NF의 단점을 빠트리는데 그 이유는 4NF를 위반하는 테이블(그렇지만 모든 하위 정규형을 만족시키는)은 거의가 업무 상 이루어지지 않는다는 믿음 때문이라 하였다. 이러한 믿음은 그렇지만 정확하지 않을 수도 있다. Wu는 40개의 기관 데이터베이스 연구에서, 20% 이상이 모든 하위 정규형을 만족시키고 있지만 4NF를 위반하고 있는 하나 이상의 테이블을 갖고 있다는 것을 밝혔다.

8.4.7 Fifth normal form

5NF는 또한 project-join normal form (PJ/NF)라고도 하며, 어의적으로 관련된 복수의 관련성을 고립시킴으로써 복수값 사실(multi-valued facts)을 레코딩하는 관계형 데이터베이스에서 과잉을 줄이기 위해 디자인된 데이터베이스 정규화의 한 단계이다. 만일 그리고 단지 모든 non-trivial join dependency이 후보키들에 의해 수반된다면(implied), 그 테이블은 5NF에 있다고 말한다.

R에 대한 join dependency *{A, B, … Z}가 만일 그리고 단지 A, B, …, Z의 각각이 R을 위한 a superkey일 때만 R의 후보키(들)에 의해 수반(implied)된다.

8.4.7.1 Example

Traveling Salesman Product Availability By Brand

Traveling Salesman	Brand	Product Type
Jack Schneider	Acme	Vacuum Cleaner
Jack Schneider	Acme	Breadbox
Willy Loman	Robusto	Pruning Shears
Willy Loman	Robusto	Vacuum Cleaner
Willy Loman	Robusto	Breadbox
Willy Loman	Robusto	Umbrella Stand
Louis Ferguson	Robusto	Vacuum Cleaner
Louis Ferguson	Robusto	Telescope
Louis Ferguson	Acme	Vacuum Cleaner
Louis Ferguson	Acme	Lava Lamp
Louis Ferguson	Nimbus	Tie Rack

이 테이블은 Brand로 설정된 상표에 의해 만들어진 Product Type과 이것의 종류를 나타내는 Products는 Traveling Salesman으로 설정된 판매원으로부터 이용할 수 있다는 것을 말하고 있다.

Traveling Salesman, Brand, and Product Type의 유효한 결합을 제한하는 어떤 규칙의 부재로 인하여, 이 3-속성 테이블은 이러한 상황을 정확하게 모델화할 필요가 있다.

그렇지만, 아래의 규칙을 적용한다고 가정해 보자: A Traveling Salesman이 자신의 repertoire에 certain Brands and certain Product Types을 갖고 있다. 만일 Brand B1과 B2가 그의 레포터리에 있다면, 그리고 Product Type P도 그의 레포터리에 있다면 (Brand B1 and Brand B2 둘 다 Product Type P를 만든다고 가정한다면), Traveling Salesman 은 products of Product Type P those made by Brand B1 and those made by Brand B2 를 제공해야만 한다.

이러한 경우에, 이 테이블은 3개로 나눌 수 있다:

Product Types By Traveling Salesman

Traveling Salesman	Product Type
Jack Schneider	Vacuum Cleaner
Jack Schneider	Breadbox
Willy Loman	Pruning Shears
Willy Loman	Vacuum Cleaner
Willy Loman	Breadbox
Willy Loman	Umbrella Stand
Louis Ferguson	Telescope
Louis Ferguson	Vacuum Cleaner
Louis Ferguson	Lava Lamp
Louis Ferguson	Tie Rack

Brands By Traveling Salesman

Traveling Salesman	Brand
Jack Schneider	Acme
Willy Loman	Robusto
Louis Ferguson	Robusto
Louis Ferguson	Acme

Louis Ferguson	Nimbus

Product Types By Brand

Brand	Product Type
Acme	Vacuum Cleaner
Acme	Breadbox
Acme	Lava Lamp
Robusto	Pruning Shears
Robusto	Vacuum Cleaner
Robusto	Breadbox
Robusto	Umbrella Stand
Robusto	Telescope
Nimbus	Tie Rack

이런 경우에, 만일 그가 ACME에 만든 그 밖의 다른 것을 팔고 또한 다른 브랜드(Robusto)의 진공청소기도 판다면, Louis Ferguson은 ACME(ACME에서 진공청소기를 만든다는 가정 하에)에서 만든 Vacuum Cleaners를 제공하는 것을 거부할 수 없다.

주목할 것은 이러한 설정(setup)이 과잉을 제거하는데 도움을 준다는 것이다. Jack Schneider가 Robusto's products Breadboxes and Vacuum Cleaners를 팔기 시작했다고 가정해 보자. 이전 설정에서, 우리는 각각의 제품 종류용의 하나로 2개의 새로운 항목을 추가해야할 것이다: <Jack Schneider, Robusto, Breadboxes>, <Jack Schneider, Robusto, Vacuum Cleaners>. 새로운 설정에서 우리는 Brands By Traveling Salesman에 있는 단일 항목(<Jack Schneider, Robusto>)만을 추가할 필요가 있다.

8.4.7.2 Usage

단지 매우 드문 상황에서만 4NF 테이블이 5NF에 따라가지는(conform) 않는다. 이러한 것들은 4NF 테이블에 있는 속성들의 값을 유효하게 결합하는데 영향을 끼치는 복잡한 실-세계의 제한조건(a complex real-world constraint)이 그 테이블의 구조에 암시적(implicit)이지 않은 상황들이다. 만일 그러한 테이블이 5NF로 정규화되지 못한다면, 그 테이블 내에서 데이터의 논리적 일관성을 유지하는데 따르는 부담을 삽입, 삭제, 갱신에 책임을 지는 어플이 부분적으로 맡아야 한다; 그리고 그 테이블에 있는 데이터가 불일치한다는 높은 위험이 존재하게 된다. 대조적으로, 5NF 디자인에서는 그러한 불일치성(inconsistencies)의 가능성을 배제시킨다.

8.4.8 DKNF(Domain/key normal form)

Domain/key normal form (DKNF)은 domain constraints와 key constraints보다 다른 어떠한 제한요소(constraints)도 없는 데이터베이스를 요구하는 데이터베이스 정규화에서 사용되는 정규형이다. domain constraint는 특정한 속성용으로 허락 가능한 값을 특정화하며, 반면에 key constraint는 특정한 테이블에 있는 로우를 유일하게 식별할 수 있는 속성을 특정화 한다.

domain/key normal form은 관계에 있는 모든 제한요소가 keys와 domains의 정의에 대

하여 논리적인 결과일 때 완성되며, key와 domain restraint(금지, 제지)와 조건을 강화함으로써 모든 제한요소를 충족시키는 원인이 된다. 그러므로 이것은 모든 non-temporal anomalies를 피한다. domain/key normal form을 사용하는 이유는 데이터베이스에서 깨끗한(clear) domain 이나 key constraints가 아닌 일반적인 제한요소를 갖는 것을 피하기 위함이다. 대부분의 데이터베이스는 속성과 관련해서 쉽게 domain and key constraints를 테스트할 수 있다. 그렇지만 일반적인 제한요소는 운영하는데 비용이 많이 들고 유지하는데도 비용이 많이 드는 stored procedures의 형태로 프로그램을 짜는 특별한 데이터베이스에서 정상적으로 요구할 것이다. 그러므로 일반적 제한요소는 domain and key constraints로 나눈다.

수 많은 이상을 품고 있을지도 모를 보다 작은 데이터베이스를 변환(convert)하는 것보다 domain/key normal form으로 데이터베이스를 구축하는 것이 훨씬 쉽다. 그렇지만 성공적으로 domain/key normal form database를 구축하는 것은 비록 경험많은 데이터베이스 프로그래머라 할 지라도 어려운 업무이다. 그러므로 domain/key normal form이 대부분의 데이터베이스에서 발견되는 문제를 제거하는 동안, 그것을 완성하는데는 가장 값비싼 정규형일 수 있다. 그렇지만 domain/key normal form을 완성하지 못하는 것은 시간이 지나면서 보다 낮은 정규형에만 매달리는 데이터베이스에 나타나는 이상들로 인하여 장기간의 걸쳐 숨어있는 비용을 처리하게 될 것이다.

3NF, Boyce-Codd NF, 4NF, 5NF는 domain/key normal form의 특별한 경우들이다. 모두가 (super) 키로 변환이 가능한 functional, multi-valued or join dependencies을 가지고 있다. 이들 정규형의 domains는 제한을 받지 않으므로, 모든 domain 제한요소들을 만족시킨다. 그렇지만 보다 고차원의 정규형을 domain/key normal form으로 변형시키는 것은 항상 dependency-preserving transformation인 것은 아니므로 항상 가능하지도 않다.

8.4.8.1 Example

다음의 테이블에서는 DKNF의 위반이 발생하고 있다:

Wealthy Person

Wealthy Person	Wealthy Person Type	Net Worth in Dollars
Steve	Eccentric Millionaire	124,543,621
Roderick	Evil Billionaire	6,553,228,893
Katrina	Eccentric Billionaire	8,829,462,998
Gary	Evil Millionaire	495,565,211

Wealthy Person의 도메인이 미리 정의해 놓은 부자의 샘플에 있는 모든 부자의 이름으로 구성되어 있고; Wealthy Person Type의 도메인은 'Eccentric Millionaire', 'Eccentric Billionaire', 'Evil Millionaire', and 'Evil Billionaire'로 구성되어 있으며; Net Worth in Dollars 도메인은 1백만보다 많거나 같거나 모두 정수로 구성되어 있다고 가정해 보자.

비록 우리가 다른 것으로부터 하나를 추론(deduce)할 수 없을 지라도Wealthy Person Type을 Net Worth in Dollars에 링크시키는데 제한(constraint)이 발생한다. 이 제한이 지적하는 것은 an Eccentric(괴짜) Millionaire or Evil Millionaire는 1,000,000에서

9099,999,999까지의 순자산(net worth)을 가질 수 있지만, 어떤 Eccentric Billionaire or Evil Billionaire는 1,000,000,000 이상의 순자산을 가질 수 있다. 이 제한요소는 domain constraint도 아니고 key constraint도 아니다: 그러므로 우리는 불일치하는 Wealthy Person Type / Net Worth in Dollars 결합이 데이터베이스에서 이루어지지 않도록 하기 위하여 domain constraints and key constraints 에 의존할 수 없다.

DKNF 위반은 단지 2개의 값 'Evil' and 'Eccentric'로 구성되는 Wealthy Person Type 도메인을 변경함으로써 제거할 수 있다: a millionaire or billionaire와 같은 부자의 지위(status)는 their Net Worth in Dollars에 은연중에 나타나 있으므로, 어떠한 유용한 정보도 잃어버리지 않는다.

Wealthy Person

Wealthy Person	Wealthy Person Type	Net Worth in Dollars
Steve	Eccentric	124,543,621
Roderick	Evil	6,553,228,893
Katrina	Eccentric	8,829,462,998
Gary	Evil	495,565,211

Wealthiness Status

Status	Minimum	Maximum
Millionaire	1,000,000	999,999,999
Billionaire	1,000,000,000	999,999,999,999

DKNF는 실재로 완성하기가 종종 어렵다.

8.4.9 Sixth normal form

6NF은 2가지 서로 다른 방법으로 사용되며, 관계형 데이터베이스 이론에 있는 전문용어(term)이다.

8.4.9.1 6NF (C. Date's definition)

temporal databases에 관한 Christopher J. Date and others의 책에서 관계형 대수(relational algebra)의 연장선상에서(based on an extension) 데이터베이스용 정규형으로 6NF를 정의하였다.

이 책에서 관계형 연산자들 - 예, join -은 날짜의 연속성 또는 시간의 한 순간과 같은 interval data의 자연스런 처리를 위하여 일반화되었다. 그런 다음에 6NF는 이러한 일반화된(generalized join)을 다음처럼 기초로 삼았다:

만일 그리고 단지 그것이 결코 어떠한 nontrivial join 의존성도 만족시키지 못한다면, relvar R [table]은 6NF에 있다 - 이전처럼, 만일 그리고 단지 적어도 관련된 projections 중에서 하나(아마도 U_projections)가 관련된 relvar [table]의 모든 속성들의 집합을 taken over하면서 a join 의존성이 trivial인 경우.

6NF에 있는 어떠한 관계라도 5NF에 있다.

6NF는 관계형 변수들을 더 이상 줄일 수 없는 구성요소들로 분해하려는 경향이 있다. 비록 이것이 non-temporal relation variables에 대하여 비교적 중요하지 않다하더라도, temporal variables or other interval data을 다룰 때 중요할 수도 있다. 예를 들어, 만일 어떤 관계가 supplier's name, status, and city로 이루어져 있다면, 우리는 또한 이러한 변수들이 현재와 과거에 유효했던(예를 들어, 역사적 데이터에 대한) 시간과 같은 일시적 데이터(temporal data)를 추가하려할 수도 있으나, 이 3개의 값이 서로 독립적으로 그리고 다른 비율로 변할 지도 모른다. 경우에 따라 우리는 Status를 변화시킨 역사를 추적하려할 지도 모른다.

8.4.9.2 DKNF

몇몇 저자들은 6NF를 다르게 즉, Domain/key normal form (DKNF)의 이음동의어로 사용한다.

8.4.9.3 Usage

6NF는 예를 들어 Anchor Modeling을 사용함으로써 현재 장점이 단점보다 많은 몇몇 data warehouses에서 사용되고 있다. 비록 6NF를 사용하는 것이 테이블의 폭발적 증가를 가져오더라도, 현대 데이터베이스는 ('table elimination'이라 부르는 과정을 사용하여) select queries로부터 요구되지 않고 따라서 여러 가지 속성만을 접근하는 쿼리의 속도를 높이기 위하여 테이블들을 제거할 수 있다(prune).

1) **Anchor Modeling** is an agile database modeling technique suited for information that changes over time both in structure and content. It provides a graphical notation used for conceptual modeling similar to that of entity-relationship modeling, with extensions for working with temporal data. The modeling technique is based around four modeling constructs: the anchor, attribute, tie and knot, each capturing different aspects of the domain being modeled. The resulting models can be translated to physical database designs using formalized rules. When such a translation is done the tables in the relational database will mostly be in the sixth normal form.

8.5 Denormalization

online transaction processing (OLTP)을 목적으로 하는 데이터베이스는 정형적으로 online analytical processing (OLAP)을 목적으로 하는 데이터베이스보다 더 정규화되어야 한다. OLTP 어플들은 슈퍼마케 카운터에서 판매 레코드를 갱신하는 것과 같은 많은 양의 작은 거래(transactions)을 할 수 있다는 특징을 가지고 있다. 이러한 기대는 각 거래가 일관된 상태로 데이터베이스에 남게 될이란 것이다. 대조적으로, OLAP operations을 목적으로 하는 데이터베이스는 우선적으로 "read mostly" 데이터베이스이다. OLAP 어플들은 어랜기간 동안 쌓여온 역사적 데이터를 추출하려는 경향을 가지고 있다. 그러한 데이터베이스와 관련해서, 과잉적이고 "탈정규화적"인 데이터는 business intelligence applications을 원활하게 할 수 있다. 특히, star schema에 있는 dimensional tables는 종종 탈정규화 데이터를 포함하고 있다. 탈정규화된 또는 과잉된 데이터는 extract, transform, load (ETL) 과정동안 주의깊게 통제되어야 하며, 이용자들은 그것이 일관된 상태(consistent state)가 될 때까지 그 데이터를 볼 수 있도록 허용해서는 안된다. star schema에 대하여 정규화된 대안은 snowflake schema 이다. 많은 사례에서, computers and RDBMS software의 파워가 점점 더 강력하게

됨으로써, 탈정규화의 필요성이 줄어들고 있다(wane). 그러나 데이터의 부피가 일반적으로 hardware and software performance에 따라 증가함으로써, OLAP 데이터베이스는 종종 아직까지 탈정규화된 스키마를 사용하고 있다.

탈정규화는 또한 computerized cash-registers and mobile devices에서 단지 look-up (예: price lookups)용 데이터만을 사용할 수도 있으므로 소형 컴퓨터의 성능을 개선하기 위하여 사용된다. 탈정규화는 또한 어떠한 RDBMS도 Palm과 같은 프랫폼용으로 존재하지 않을 때 그렇지 않으면 데이터와 swift에 책임을 져야하는 어떠한 변화도 중요하지 않을 때 사용될 수도 있다

8.5.1 Non-first normal form(NF^2 or N1NF)

탈정규화(Denormalization)는 정규화의 반대이다. 탈정규화는 정교하고 유용할 수 있다는 인식에 따라, non-first normal form은 "sets and sets of sets to be attribute domains" (Schek 1982)를 허용함으로써 1NF를 따르지 않는 데이터베이스의 디자인으로 정의되고 있다. 따라서 이 모델에서 데이터를 쿼리하고 조작하는데 사용된 언어는 그 같은 값들을 지원하도록 확대되어야만 한다.

이것을 살펴보는 한 가지 방법은 그러한 구조의(structured) 값들을 스스로의 도메인-전문(domain-specific) 언어를 갖고 있는 특정화된 유형의 값들(domains)로 고려하는 것이다. 그렇지만, non-1NF 모델이 대체로 의미하는 것은 관계형 모델과 그것에 쿼리하기 위하여 사용된 언어가 그러한 구조에 맞게 일반적인 메카니즘으로 확대되는 approach이다: 예를 들어 the nested(둥지) relational model은 2개의 추가적인 연산자((*nest* 와 *unnest*)를 별도로 nested relations를 만들고 flatten(평평하게 펴다)할 수 있는 관계형 대수(algebra)에 추가함으로써, 도메인 값처럼 관계들의 사용을 지원한다.

아래의 테이블을 살펴보자:

Person	Favourite Colour
Bob	blue
Bob	red
Jane	green
Jane	yellow
Jane	red

어떤 사람이 좋아하는 여러 가지의 색깔이 있다고 가정해 보자. 분명하게도, 선호하는 색깔들은 이 테이블에 의해 모델화된 한 세트의 색깔들로 이루어져 있다. 1NF를 NF^2 테이블로 변형시키기 위하여, 보다 높은 수준의 정규형의 관계형 대수로 확장시키는 “nest” 연산자가 필요하다. “nest”연산자를 1NF 테이블에 적용시키는 것은 다음과 같은 NF^2 테이블에서 가능하다:

Person	Favourite Colours
Bob	Favourite Colour blue

	red
Jane	Favourite Colour green yellow red

이러한 NF^2 테이블을 다시 1NF로 변형시키기 위해서는 보다 높은 수준의 정규형의 관계형 대수를 확장시키는 “unnest“ 연산자가 필요하다. 위의 경우에, unnest는 ”colours“를 자신의 테이블에 만들 수 있다. 비록 ”unnest“가 ”nest“에 대한 수학적 반대(inverse)라 하더라도, ”nest“ 연산자가 항상 ”unnest“의 수학적 반대만은 아니다. 필요한 또다른 제한요소는 그 연산자가 the Partitioned Normal Form (PNF)에 의해 covered되는 bijective 여야 한다는 것이다.

9. Performance, security, and availability

기업의 원활한 운영을 위하여 데이터베이스 기술은 매우 중요하기 때문에, 데이터베이스 시스템에는 필요한 성능, 보안성, 그리고 이용성(performance, security, and availability)과 관련된 복잡한 메카니즘이 포함되어 있으며, 데이터베이스 운영자로 하여금 이러한 특징의 사용을 제어하도록 한다.

9.1 Database storage

데이터베이스 기억장치(storage)란 데이터베이스의 물리적 형체(materialization)의 container 다. 이것은 데이터베이스 구조의 내적(물리적) 수준(internal (physical) level)을 형성한다. 또한 이것은 필요할 때 그 내적 차원으로부터 개념적 수준과 외적 수준(conceptual level and external level)을 재구축하는데 필요한 모든 정보(예, "데이터의 데이터"인 메타데이터와 내적 데이터 구조들)를 포함하고 있다. 데이터를 항구적인 기억장치에 넣은 것은 일반적으로 그 데이터베이스 엔진 즉, (a.k.a.:also known as, 별칭은, 별명은) "storage engine"의 책임이다. 기본 운영체제를 통하여 DBMS에 접근하는 것이 전형적이라 하더라도(그리고 종종 storage layout를 위한 매개체로서 운영체제의 파일 시스템을 활성화시키더라도), 기억장치의 성질과 구성은 그 DBMS의 효율적인 운영을 위해 극히 중요하므로 데이터베이스 행정가에 의해 면밀하게 관리되고 있다. DBMS는 현재 운영 중이더라도 여러 유형의 기억장치(예, 메모리와 외적 저장고)가 딸려있는 자기만의 데이터베이스를 늘 가지고 있다. 데이터베이스의 데이터와 추가적으로 필요한 정보는 아마도 매우 많을 것이지만, 이것들은 bits로 암호화 되어있다. 전형적으로 데이터는 개념적 그리고 외적 수준의 데이터를 찾는 방식과는 완전히 다르게 보이는 구조의 기억장치에 들어 있으나, 데이터로부터 필요한 정보의 추가적 유형을 계산하기 위하여 (예, 데이터베이스에 쿼리가 발생할 때), 그 뿐만 아니라 이용자와 프로그램에 의해 요구될 때, 이러한 수준들의 재구성을 최적화시킬 수 있는 방법에 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.

어떤 DBMS는 특정한 문자암호화를 사용하여 데이터를 저장하도록 하므로, 여러 가지 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

여러 가지 low-level(저급한) 데이터베이스 기억장치 구조는 데이터 모델을 연속화(serialize)할 수 있는 기억장치 엔진에서 사용되므로, 선택된 매체에 맞춰 작성(written) 될 수 있다. 색인화와 같은 기법이 성능개선을 위해 사용될 수 있다. 전통적인 기억장치는 열-지향적(row-oriented),이만 행-지향적이면서 상호관계적인(column-oriented and correlation) 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

9.1.1 Database materialized views

가끔 기억의 여분(storage redundancy)이 성능을 높이기 위하여 사용된다. 일반적인 한 가지 사례는 때때로 필요로 하는 external views나 쿼리 결과로 구성되는 사실 뷰(materialized views)를 저장하는 것이다. 그러한 뷰를 저장하는 것은 그것들이 필요할 때마다 그것들을 처리하는데 드는 비싼 비용(expensive computing)을 절약한다. 사실 뷰의 단점(downsides)은 최초로 갱신된 데이터베이스 데이터와 그것들을 동기화시키고자 갱신할 때 발생하는 추가비용(overhead)과 기억의 여분에 대한 경비(cost)이다.

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.

9.1.2 Database and database object replication

경우에 따라서, 데이터베이스는 (동일한 데이터베이스 사물에 다수의 최종이용자가 동시에 접근할 수 있도록 성능을 개선하고, 분산형 데이터베이스의 부분적 잘못에 대해서는 회복력(resiliency)을 제공하는 두 가지 모두에 대한) 데이터의 이용성을 높이기 위하여, (하나이상의 사본과 함께) 데이터베이스 사물 복제(database objects replication)를 통해 기억의 여분을 사용한다. 복제된 사물의 갱신은 그 사물의 사본 간에 동시에 이루어져야 한다. 많은 경우에서 전체 데이터베이스를 복제한다.

9.2 Database security

데이터베이스 보안이란 데이터베이스의 콘텐트, 소유자, 이용자를 보호하는 모든 요소를 다루는 것이다. 이것의 범위는 고의적이고 불법적인 데이터베이스 사용으로부터의 보호에서부터 불법 객체(예, 사람이나 컴퓨터 프로그램)에 의한 비고의적인 접근까지이다.

데이터베이스 접근 통제란 누가(사람이나 어떤 컴퓨터 프로그램) 데이터베이스에 있는 어떤 정보에 접근할 수 있는지를 다루는 것이다. 여기서 정보는 (예, 레코드 종류, 특수 레코드, 데이터 구조와 같은) 특별한 데이터베이스 사물, (예, 쿼리 유형 또는 특별한 쿼리와 같은) 어떤 사물에 대한 어떤 computations, 또는 (예, 특수한 색인이나 정보접근을 위한 기타 데이터 구조를 이용하는 것과 같은) 전자(the former)로의 특별한 접근로의 활성화에 관한 것들일 수 있다. 데이터베이스 접근 통제는 전용 보호 안전용(dedicated protected security) DBMS 인터페이스를 사용하며 (데이터베이스 소유자에 의해) 특별한 권한을 부여받은 요원에 의해 설정된다.

이것은 개별적 근거에 따라, 또는 개인과 특권을 집단에 양도함으로써, 또는 (대부분의 정교한 모델에서) 자격(entitlement)이 필요한 역할에 따라 개인과 집단을 배정함으로써 직접적으로 관리되기도 한다. 데이터 보안은 불법이용자가 데이터베이스를 살펴보거나 갱신하는 것을 예방한다. 패스워드를 사용함으로써, 이용자는 데이터베이스 전체와 소위 "subschemas"라 부르는 부분집합에 접근할 수 있다. 예를 들어, 직원 데이터베이스는 직원 개인에 대한 모든 데이터를 포함되어 있지만, 어떤 이용자 집단은 단지 급료 데이터만을 볼 수 있는 자격이 있는 반면에 또 다른 집단은 작업이력과 의료 데이터에만 접근할 수 있다. 만일 DBMS가 데이터베이스의 입력과 갱신뿐만 아니라 그것에 질문하는(interrogate) 쌍방향성을 제공한다면, 이러한 기능을 통하여 인사 데이터베이스를 관리할 수 있다.

데이터 보안이란 일반적으로 데이터의 특정한 chunks(규모)를, 물리적으로(다시 말해서, 부패, 파괴, 제거로부터; 예, 물리적 보안을 참조할 것) 또는 데이터의 전부 또는 일부를 의미 있는 정보로 해석한 것(예, 포함하고 있는 일련의 비트를 살펴봄으로써, 특정하고 유효한 신용카드 번호라고 결론짓는 것; 예, 데이터 암호화를 참조할 것) 둘 다를 다루는 것이다.

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.

*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.

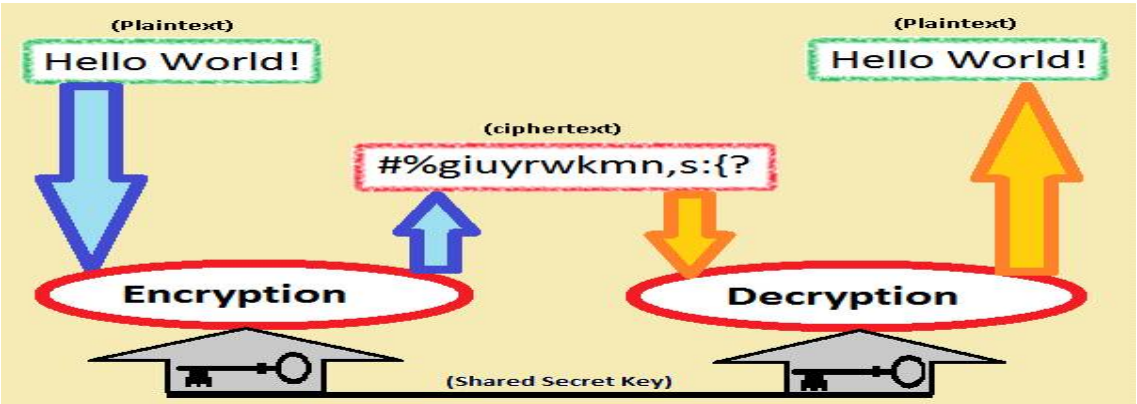
*Security personnel

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 logging에는 누가 어떤 속성에 접근해서 무엇을 바꾸었고, 언제 그런 변화가 이루어졌는지를 기록하고 있다. 로깅 서비스는 접근의 발생과 변화에 대한 기록을 유지

함으로써 나중에 forensic database audit(부검과 같은 데이터베이스 감사)를 할 수 있도록 한다. 때때로 application-level code가 데이터베이스를 위해 이러한 흔적을 지우기(leaving)보다는 변화를 기록하는데 사용된다. 모니터링은 보안 위반(breaches)을 탐지하기 위하여 설치될 수 있다.

9.3 Transactions and concurrency

데이터베이스 거래(transactions)는 어떤 파괴(crash)로부터 회복된 다음에, fault tolerance와 데이터 순수성(data integrity)의 수준을 어떻게 맞출 것인가(introduce)와 관련해서 사용될 수 있다. 데이터베이스 거래는 전형적으로 데이터베이스의 다양한 운영(예, 데이터베이스의 사물을 읽고, 쓰고, lock를 수집하는 것 등)에 포함되면서, 데이터베이스와 다른 시스템에서 지원하고 있는 하나의 추상(abstraction)과 같은 작업의 한 단위(unit)이다. 각각의 거래는 그 거래에(특별한 거래 명령어를 통하여 거래 프로그래머에 의해 결정된) 어떠한 프로그램/코드의 실행이 포함 되어 있는지와 관련된 잘 정의된 영역(boundaries)을 갖고 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.

ACID란 데이터베이스 거래의 이상적인 성질을 설명하고 있다: 원자값, 일관성, 독립성, 인내성(Atomicity, Consistency, Isolation, and Durability).

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.

9.4 Migration

하나의 DBMS와 함께 만든 데이터베이스는 다른 DBMS로 운반(portable)할 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 (서로 다른 DBMS는 서로 다른 TCOs를 가질 수 있기 때문에) 경제적으로, 기능적으로, 그리고 운영상(서로 다른 DBMSs는 서로 다른 성능을 가질 수 있다) 필요하기 때문이다. 이런 변형이 만일 가능하다면, 데이터베이스와 관련된 어플(다시 말해서 관련된 모든 어플 프로그램) 역시 손대지 않고 가능해야 한다. 따라서 데이터베이스의 개념적 그리고 외적 구조 수준은 그러한 변형 안에서도 유지되어야 한다. 또한 어떤 구조들은 내적 수준을 계속적으로 유지하길 원할 수도 있다. 복잡하거나 대형인 데이터베이스의 이주(migrations)는 그 자체로 복잡하고 값비싼 (일회성) 프로젝트일 수 있으며, 이주 결정 시에 이 점을 고려하여야 한다. 이러한 사실에도 불구하고 특별한 DBMS 간의 이주를 지원하는 많은 도구가 존재하고 있으며, 전형적으로 DBMS vendor는 다른 인기 있는 DBMS로부터 데이터베이스를 importing(수입)하는 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.

9.5 Database building, maintaining, and tuning

어플용 데이터베이스를 디자인한 다음에, 그 다음 단계는 데이터베이스를 구축하는 것이다. 전형적으로 올바른 범용 DBMS는 이런 목적을 실현하기 위하여 선택될 수 있다. DBMS는 그것의 각 데이터 모델에서 필요한 어플의 데이터 구조를 정의할 수 있도록 데이터베이스 관리자에 의해 활용하는데 필요한 이용자 인터페이스를 제공하고 있다. 다른 이용자 인터페이스는 필요로 하는 DBMS의 parameters(보안관련, 기억할당량 매개변수 등)를 선택하는데 이용된다.

데이터베이스가 준비되면(모든 그것의 데이터 구조와 기타 필요한 구성요소가 정의되면), 그것을 운영하기 전에 전형적으로 초기용(initial) 어플의 데이터(전형적으로 하나의 분명한 프로젝트인 데이터베이스 초기화; 많은 경우에 대량의 입력을 지원하는 전문화된 DBMS 인터페이스의 사용)를 갖추어야(populated) 한다. 어떤 경우에 데이터베이스는 그러한 어플 데이터 없이 운영이 되며, 데이터는 운영하는 동안 축적된다.

데이터베이스가 만들어지고, 초기화되고, 필요한 데이터가 갖추어진 다음, 그것은 유지 관리되어야 한다. 여러 가지 데이터베이스 변수를 변경함으로써 그 데이터베이스는 보다 높은 성능용으로 튜닝되기도 한다; 어플의 데이터 구조는 변하거나 추가될 수 있으며, 새롭게 관련된 어플 프로그램이 그 어플의 기능성 등에 추가되도록 만들어질 수 있다. 데이터베이스는 종종 Microsoft Excel과 같은 spreadsheets와 혼돈된다. Microsoft Access는 데이터베이스 관리 시스템이며, Excel은 스프레드쉬트 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.

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

9.6 Backup and restore

때때로 여러 가지 이유로 이전 상태로 되돌려 놓아야 하는 경우가 발생한다. 예를 들어, 데이터베이스가 소프트웨어 에러로 인하여 문제가 발생하거나 잘못된 데이터로 갱신을 한 경우이다. 이러한 것을 방지할 목적으로, 각각의 데이터베이스 상태(다시 말해서, 데이터베이스의 데이터의 값들과 데이터베이스의 데이터 구조에 그것들의 포함(embedding))가 전용 백업 파일을 통해 유지 관리될 수 있도록, 백업 기능이 필요에 따라 또는 지속적으로 작용하고 있다. 이러한 것들을 효과적으로 할 수 있는 많은 기술들이 존재하고 있으며, 이러한 상태가 필요할 때, 다시 말해서 데이터베이스 관리자가 이러한 상태로 데이터베이스를 되돌려 놓기로 결정했을 때(예를 들어, 데이터베이스가 전에 이런 상태에 있었을 시점의 상태를 특정화함으로써), 이 파일들은 그러한 상태로 회복(restore)시키는데 활용된다.

9.7 Other

기타 DBMS의 특징으로는 다음과 같은 것이 있다:

1) Database logs
2) Graphics component for producing graphs and charts, especially in a data warehouse system
3) Query optimizer – Performs query optimization on every query to choose 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.

4) Tools or hooks for database design, application programming, application program maintenance, database performance analysis and monitoring, database configuration monitoring, DBMS hardware configuration (a DBMS and related database may span computers, networks, and storage units) and related database mapping (especially for a distributed DBMS), storage allocation and database layout monitoring, storage migration, etc.

< 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
19) "OWL DL Semantics". Retrieved 10 December 2010.
20) itl.nist.gov (1993) Integration Definition for Information Modeling (IDEFIX). 21 December 1993.
21) Chapple, Mike. "SQL Fundamentals". Databases. About.com. Retrieved 2009-01-28.
22) "Structured Query Language (SQL)". International Business Machines. October 27, 2006. Retrieved 2007-06-10.
23) Wagner, Michael (2010), "1. Auflage", SQL/XML:2006 - Evaluierung der Standardkonformität ausgewählter Datenbanksysteme, Diplomica Verlag, ISBN 3-8366-9609-6

< 부 록 >

*** List of Academic Databases and Search Engines ***

<Name; Discipline(s); Description; Provider(s)>

* **Academic Search;**
  Multidisciplinary
  Several versions: Complete, Elite, Premier, and Alumni Edition;
  EBSCO Publishing

* **Aerospace & High Technology Database**
  Aerospace, Aeronautics, Astronautics
  ProQuest

* **African Journals OnLine (AJOL)**
  Multidisciplinary
  Scholarly journals published in Africa Free abstracts;
  African Journals OnLine.

* **AgeLine**
  Sociology, Gerontology
  Includes information on aging-related topics, including economics, public health and policy.
  EBSCO Publishing.

* **AGRICOLA: Agricultural Online Access**
  Agriculture
  Produced by the United States National Agricultural Library.
  Free access provided by NAL.
  Subscription access provided by Proquest, OVID.

* **AGRIS: Agricultural database**
  Agriculture
  Covers agriculture, forestry, animal husbandry, aquatic sciences and fisheries, human nutrition, extension literature from over 100 participating countries. Material includes unique grey literature such as unpublished scientific and technical reports, theses, conference papers, government publications, and more.
  Produced by the Food and Agriculture Organization of the United Nations.
  http://agris.fao.org AGRIS

* **Airiti Inc.**
  Multidisciplinary
  China, Taiwan. Airiti Inc.

* **Analytical Abstracts**
  Chemistry
  Royal Society of Chemistry

* **Analytical sciences digital library**
  Analytical chemistry
  National Science Digital Library and the Analytical Chemistry Division of the American Chemical Society

* **Anthropological Index Online**
  Anthropology Index only (no abstracts or full-text)
  Royal Anthropological Institute

* **Anthropological Literature**
  Anthropology
  Maintained by Harvard University. Non-Harvard access provided by OCLC

* **Arachne**
  Archaeology, Art history
  German Archaeological Institute & the University of Cologne

* **Arnetminer**
  Computer Science
  Online service used to index and search academic social networks
  Tsinghua University

* **Arts & Humanities Citation Index**
  Arts, Humanities Part of Web of Science
  Thomson Reuters

* **arXiv**
  Physics, Mathematics, Computer science, Nonlinear sciences, Quantitative biology and Statistics
  Cornell University

* **Association for Computing Machinery Digital Library**

Computer Science, Engineering,
Association for Computing Machinery.

* **Astrophysics Data System**
Astrophysics, Geophysics, Physics,
Harvard University.

* **ATLA Religion Database**
Religious studies,
Provides information on topics such as biblical studies, world religions, church history, and religion in social issues.

* **AULIMP: Air University Library's Index to Military Periodicals**
Military Science,
Air University.

* **BASE: Bielefeld Academic Search Engine**
Multidisciplinary,
Bielefeld University.

* **Beilstein database**
Organic chemistry,
Available from Elsevier under the product name Reaxys.

* **Biological Abstracts**
Biology,
A complete collection of bibliographic references covering life science and biomedical research literature published from more than 4,000 journals internationally,
Available from Thomson Reuters.

* **BioOne**
Biology, Ecology, and Environmental Science
An aggregation of over 78,000 peer-reviewed, full-text articles on current research in Biodiversity Conservation, Biology, Ecology, Plant Sciences, Entomology, Ornithology, and Zoology. Free Abstract & References, Subscription Collections, and an Open Access Collection Available from BioOne.

* **Bioinformatic Harvester**
Biology, Bioinformatics A meta search engine for 50 major bioinformatic databases and projects.

Free Available from Liebel-Lab, KIT Karlsruhe Institute of Technology.

* **Book Review Index Online**
  Book reviews,
  Thomson Gale

* **Books In Print**
  Books,
  R.R. Bowker

* **CAB Abstracts**
  Applied Life Sciences Bibliographic information service providing access to applied life sciences literature,
  CABI.

* **Chemical Abstracts Service**
  Chemistry,
  American Chemical Society.

* **ChemXSeer**
  Chemistry,
  Pennsylvania State University.

* Chinese Social Science Citation Index
  Social sciences,
  Nanjing University

* **Cochrane Library**
  Medicine, Healthcare,
  Includes reviews of research to promote evidence-based healthcare,
  Wiley Interscience.

* **CINAHL: Cumulative Index to Nursing and Allied Health**
  Nursing, Allied Health,
  EBSCO.

* **CHBD: Circumpolar Health Bibliographic Database**
  Medicine
  University of Calgary.

* **Citebase Search**

Mathematics, Computer science, Physics,
Semi-autonomous citation index of free online research,
University of Southampton.

* **CiteULike**
Computer science.

* **CiteSeer**
Computer Science,
Replaced by CiteSeerX,
Pennsylvania State University.

* **CiteSeerX**
Computer science, Statistics, Mathematics, becoming Multidisciplinary
Pennsylvania State University.

* **CogPrints: Cognitive Sciences Eprint Archives**
Science (General),
University of Southampton.

* **The Collection of Computer Science Bibliographies**
Computer science,
Alf-Christian Achilles.

* **Compendex**
Engineering Electronic version of Engineering Index,
Elsevier.

* **Current Index to Statistics**
Statistics,
Limited free search,
American Statistical Association and the Institute of Mathematical Statistics.

* **Current Contents**
Multidisciplinary,
Part of Web of Knowledge. Contains 7 discipline-specific subsets.
Thomson Reuters.

* **Directory of Open Access Journals**
Journals,
Lund University.

* DBLP

Computer science,

Comprehensive list of papers from major computer science conferences and journals,

University of Trier, Germany

* EconBiz

Economics,

EconBiz supports research in and teaching of economics with a central entry point for all kinds of subject-specific information and direct access to full texts.

Produced by the ZBW- German National Library of Economics- Leibniz Information Centre for Economics (ZBW).

* EconLit

Economics,

The American Economic Association's electronic database, the world's foremost source of references to economic literature.

the American Economic Association. Available from CSA, DIALOG, OCLC, OVID, and AEA.

* EMBASE

Biomedicine, Pharmacology,

Biomedical database with a strong focus on drug and pharmaceutical research. Elsevier.

* ERIC: Educational Resource Information Center

Education,

Education literature and resources. Provides access to over 1.3 million records dating back to 1966.

the United States Department of Education. Also available by subscription from OCLC, CSA.

* Food Science and Technology Abstracts

Food science, Food technology, Nutrition

The world's leading database of information on food science, food technology and nutrition.

the International Food Information Service. Access provided by OVID, Web of Knowledge, Dialog, DataStar and STN International.

* GENESIS

Women's history,

Descriptions of women's history collections from sources in the UK, as well as women's history websites.

London Metropolitan University.

* Global Health

Public Health,

Specialist bibliographic, abstracting and indexing datatabase dedicated to public health research and practice.

CABI.

* Google Scholar

Multidisciplinary,

Google.

* GoPubMed

Medicine,

GoPubMed, the first knowledge-based search engine for the life sciences industry.

Transinsight.

* HubMed

Medicine,

An alternative interface to the PubMed medical literature database.

Alf Eaton.

* IEEE Xplore

Computer Science, Engineering, Electronics,

IEEE.

* Index Copernicus

Multidisciplinary science,

Scientific journal database – the IC Journal Master List – contains currently over 2,500 journals from all over the world, including 700 journals from Poland.

The journals registered in this database underwent rigorous, multidimensional parameterization, proving high quality. The Ministry of Science and Higher Education acknowledged the IC Journal Master List by placing it on the list of scored databases, for being indexed in IC JML journals get additional points in the Ministry's evaluation process,

Index Copernicus International.

* **Information Bridge: Department of Energy Scientific and Technical Information**
  Multidisciplinary,
  The Information Bridge: DOE Scientific and Technical Information provides free public access to over 266,000 full-text documents and bibliographic citations of Department of Energy (DOE) research report literature. Documents are primarily from 1991 forward and were produced by DOE, the DOE contractor community, and/or DOE grantees. Legacy documents are added as they become available in electronic format, United States Department of Energy, Office of Scientific and Technical information

* **Informit**
  Multidisciplinary,
  Australasian aggregator of bibliographic databases and journals,
  RMIT Publishing.

* **IngentaConnect**
  Multidisciplinary,
  Free searching,
  Ingenta.

* **Indian Citation Index**
  Multidisciplinary
  Indian Citation Index (ICI) is a home grown abstracts and citation database, with multidisciplinary objective knowledge contents from about 1000 top Indian scholarly journals. It provides powerful search engine to fulfill search and evaluation purposes for researchers, policy makers, decision makers etc.,
  ICI.

* **Inspec**
  Physics, Engineering, Computer Science,
  The leading bibliographic database providing abstracts and indexing to the d's scientific and technical papers in physics, electrical engineering, electronics, unications, control engineering, computing, information technology, manufacturing, production, and mechanical engineering.
  IET

* **International Directory of Philosophy**
  Philosophy,
  Contains information on university philosophy departments and programs,

philosophical societies, research centers, journals, and philosophy publishers in the U.S., Canada, and approximately 130 other countries. Free search; full access
Philosophy Documentation Center.

* **Intute**
Multidisciplinary Serves students, teachers, and researchers in UK further education and higher education, offering a selection of around 300,000 academic websites which have been hand-picked and described by subject specialists. No longer maintained.
Intute.

* **JournalSeek**
Multidisciplinary,
Open access journals in different language,
Links to journal's home page and publishers JournalSeek.

* **JSTOR: Journal Storage**
Multidisciplinary (Historical),
JSTOR.

* **Jurn**
Multidisciplinary,
Open access journals, primarily in the arts and humanities, but also coverage in science, biomedical, and economics.
Jurn.

* **Lesson Planet**
Education (K-12),
Over 400,000 teacher-reviewed classroom resources including lesson plans, worksheets, educational videos, and education articles.
Free Abstract; Subscription full-text Lesson Planet.

* **LexisNexis**
Law (general),
Electronic database for legal and public-records related information,
Reed Elsevier.

* **MathSciNet**
Mathematics,
Available in print as Mathematical Reviews,

American Mathematical Society.

* **MedlinePlus**
  Medicine,
  Free Produced by the United States National Library of Medicine, the United States National Institutes of Health, and the United States Department of Health and Human Services.

* **Mendeley**
  Multidisciplinary,
  The Mendeley research catalog is a crowdsourced database of research documents. Researchers have uploaded nearly 100M documents into the catalog with additional contributions coming directly from subject repositories like Pubmed Central and Arxiv.org or web crawls.
  Mendeley.

* **Merck Index,**
  Chemistry, Biology, Pharmacology,
  Also available in print. Subscription Produced by Merck & Co.. Available from CambridgeSoft Corporation, Dialog, Knovel, MedicinesComplete, STN International.

* **Meteorological and Geoastrophysical Abstracts,**
  Meteorology, Astrophysics, Geology,
  the American Meteorological Society. Available from Dialog and CSA.

* **Microsoft Academic Search**
  Computer Science and a limited extent on information science,
  Provides many innovative ways to explore scientific papers, conferences, journals, and authors,
  Microsoft.

* **NBER: National Bureau of Economic Research**
  Economics,
  National Bureau of Economic Research.

* **National Criminal Justice Reference Service**
  Criminology, Sociology,
  Abstracts of scholarly journal articles, agency and NGO reports, and conference proceedings.
  United States Department of Justice, Office of Justice Programs.

* **National Diet Library Collection**
  Multidisciplinary,
  Japanese. Catalog for the National Library of Japan.
  National Diet Library.

* **OAIster**
  Multidisciplinary,
  OCLC.

* **OpenSIGLE**
  Grey literature,
  Indexes European grey literature.
  Institut de l'information scientifique et technique.

* **Philosophy Documentation Center eCollection**
  Applied ethics, Philosophy, Religious studies,
  Journals, series, conference proceedings, and other works from several countries online.
  Free abstract & preview; Subscription full-text Philosophy Documentation Center.

* **Philosophy Research Index**
  Philosophy,
  Index of books, journals, disserations, and other documents,
  Philosophy Documentation Center.

* **PhilPapers**
  Philosophy,
  PhilPapers.

* **POIESIS: Philosophy Online Serials**
  Philosophy, applied ethics, religious studies,
  Journals and series, online access for institutions with print,
  Free abstract & preview; Subscription full-text Philosophy Documentation Center.

* **POPLINE**
  Population, Family Planning, Reproductive Health,
  POPLINE® contains the world's most comprehensive collection of population, family planning and related reproductive health and development literature. An

international resource, POPLINE helps program managers, policy makers, and service providers in low- and middle-income countries and in development-supportive agencies and organizations gain access to journal articles and other scientific, technical, and programmatic publications.
Knowledge for Health, Center for Communication Programs, Johns Hopkins Bloomberg School of Public Health.

* **PsycINFO**
  Psychology,
  The largest resource devoted to peer-reviewed literature in behavioral science and mental health. It contains over 2.6 million citations and summaries dating as far back as the early 19th century.
  the APA.

* **Pubget**
  Multidisciplinary,
  Pubget.

* **PubMed**
  Biomedical,
  National Institutes of Health and the U.S. National Library of Medicine.

* **PubChem**
  Chemistry,
  National Center for Biotechnology Information and the U.S. National Library of Medicine.

* **Questia: Online Research Library**
  Multidisciplinary (Historical),
  Questia.

* **Readers' Guide to Periodical Literature**
  Journals and Magazines Coverage: 1983-present.
  H. W. Wilson.

* **Reader's Guide Retrospective: 1890-1982**
  Journals and Magazines,
  H. W. Wilson.

* **RePEc: Research Papers in Economics**
  Economics,

Volunteer Collaboration.

* **Retina medical search**

Biomedical,
Biomedical resources with special focus for medical professionals.
searches among physician level standard documents eliminating patient level materials.
Retina medical search.

* **Rock's Backpages**

Music,
Primary documents from the history of rock and roll Subscription.
Backpages Limited.

* **Russian Science Citation Index**

Scientific journals,
A bibliographic database of scientific publications in Russian.
Scientific Electronic Library.

* **SafetyLit**

Multidisciplinary,
Citations and abstracts of journal articles and reports from researchers working in the more than 35 distinct professional disciplines (architecture - zoology) relevant to preventing unintentional injuries, violence, and self-harm.
Graduate School of Public Health, San Diego State University and the World Health Organization's Department of Violence and Injury Prevention.

* **SciDiver.com**

Multidisciplinary,
SciDiver is an academic paper search engine for the physical sciences.
The service currently maintains an index over arXiv, the preprint service for mathematics, physics, astronomy, computer science, quantitative finance and related disciplines; expansion to additional repositories is expected in the course of the site's continued development.
SciDiver.com.

* **SciELO**

Journals,
SciELO is a bibliographic database and a model for cooperative electronic publishing in developing countries originally from Brazil. It contains 985 scientific journals from different countries in free and universal access,

full-text format.
FAPESP, CNPq and BIREME.

* **Science.gov**
Multidisciplinary,
A gateway to government science information and research results.
Science.gov provides a search of over 45 scientific databases and 200 million pages of science information with just one query, and is a gateway to over 2000 scientific Websites.
Science.gov Alliance, 18 scientific and technical organizations from 14 federal agencies that contribute to Science.gov. United States Department of Energy, Office of Scientific and Technical Information serves as the operating agent for Science.gov.

* **Science Accelerator**
Multidisciplinary,
A gateway to results of DOE research and development and major R&D accomplishments of interest to DOE.
United States Department of Energy, Office of Scientific and Technical Information.

* **Science Citation Index**
Science (General) Part of Web of Science,
Thomson Reuters.

* **ScienceDirect**
Multidisciplinary,
Elsevier.

* **Scirus**
Science (General),
Elsevier.

* **Scopus**
Multidisciplinary,
Elsevier.

* **SearchTeam**
Multidisciplinary,
Students search together collaboratively for scholarly articles and resources,
Zakta.

* **Social Science Citation Index**
  Social science,
  Part of Web of Science,
  Thomson Reuters.

* **Socol@r: Socolar**
  Multidisciplinary,
  Scholarly open access resources in different language
  abstracts; Links to full-text Socolar.

* **SSRN: Social Science Research Network**
  Social science,
  Contains an abstracts database and an electronic paper collection, arranged by discipline.
  Social Science Electronic Publishing, Inc.

* **SSRRN: Social Science Research Resources Network**
  Social science,
  Indexes datasets and statistical codes,
  Social Science Research Resources Network.

* **SPIRES-HEP**
  Physics, (High Energy),
  Stanford Linear Accelerator Center & partners.

* **SpringerLink**
  Multidisciplinary,
  Free abstract & preview; Subscription full-text Springer.

* **Ulrich's Periodicals Directory**
  Periodicals,
  Proquest.

* **VET-Bib**
  Social Science, Education,
  European vocational education and training (VET) literature,
  European Centre for the Development of Vocational Training.

* **Web of Knowledge**
  Multidisciplinary,

Includes other products, such as Web of Science, Biological Abstracts & The Zoological Record,
Thomson Reuters.

* **Web of Science**
Science (General),
Includes other products, such as Social Science Citation Index & Science Citation Index.
Thomson Reuters.

* **WestLaw**
Law (General),
Thomson Reuters.

* **WFL Publisher**
Food, Nutrition, Agriculture, Environment English language,
WFL Publisher & the ISFAE Ry.

* **WorldCat**
Multidisciplinary,
Unified catalog of member libraries' catalogs,
Free & Subscription OCLC.

* **WorldWideScience**
Multidisciplinary,
WorldWideScience is a global science gateway composed of national and international scientific databases and portals. WorldWideScience accelerates scientific discovery and progress by providing one-stop searching of databases from around the world. Multilingual WorldWideScience provides real-time searching and translation of globally dispersed multilingual scientific literature. The WorldWideScience Alliance, a multilateral partnership, consists of participating member countries and provides the governance structure for WorldWideScience. United States Department of Energy, Office of Scientific and Technical Information serves as the operating agent for WorldWideScience.

* **Zasshi Kiji Sakuin: Japanese Periodicals Index**
Journals,
Japanese.
National Diet Library's Online Catalog, MagazinePlus, CiNii.

* **Zentralblatt MATH**

Mathematics,
First three records free without subscription.
Springer Science+Business Media.

* **The Zoological Record**

Zoology,
Unofficial register of scientific names & papers in Zoology. Coverage 1864-present.
Thomson Reuters. :)

---------- FIN ----------