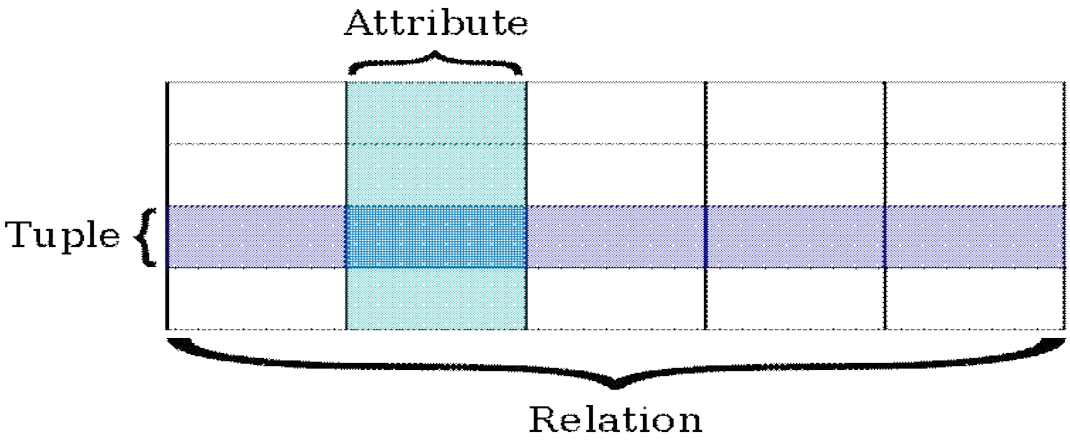
Database



2016

대구대 사회과학대학 문헌정보학과

CONTENTS

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)
a relational database management system (RDBMS)에 저장된 데이터를 관리하도록 설계된 a special-purpose programming language.

2) ODBC (Open Database Connectivity)
database management systems (DBMS)에 접근하기 위한 a standard programming language middleware API 이다.

3) JDBC
Oracle Corporation에서 개발한 a Java-based data access technology (Java Standard Edition platform)이며, 이 기술은 클라이언트가 데이터베이스에 접근하는 방법을 정의하고 있는 an API for the Java programming language 이다.

4) Application Programming Interface (API)
이것에서는 소프트웨어 구성요소들이 서로 어떻게 상호작용하여야 하는지를 밝히고 있다. 하드 디스크 드라이브나 비디오 카드 같은 데이터베이스나 컴퓨터 하드웨어에 접근뿐만 아니라, API는 graphical user interface의 인터페이스 구성요소들을 프로그래밍하는 작업을 쉽게 하는데 사용된다.

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**
데이터 잉여성과 성능개선의 목적으로 복수의 디스크 구성요소들을 하나의 논리적 unit로 결합시키는 저장 기술이다.
2) the kernel
소프트웨어에서 온 input/output requests를 관리하여 그것들은 컴퓨터의 기타 전자 구성요소와 CPU를 위한 데이터 처리 지시로 번역해 주는 컴퓨터 프로그램이다.

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**
사물이나 자료의 위치를 파악하고 관리하는 절차이다. 현대 인벤토리 통제 시스템은 종종 barcodes 그리고 radio-frequency identification (RFID) tags를 사용하여 인벤토리 사물의 자동식별기능을 제공하고 있다.

2) **FileMaker Pro**
Apple Inc.의 자회사인 FileMaker 회사에서 개발한 cross-platform relational database application 이며, 데이터베이스 엔진과 GUI-based interface를 통합하여 layouts, screens, or forms. 속으로 새로운 요소들을 dragging함으로써 이용자가 그 데이터베이스를 변경할 수 있도록 한 어플이다.

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 인터페이

스를 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
데이터가 한 포인트에서 다른 포인트로 가는 방식을 말하며, 만일 한 개 이상의 어플이 상호 운영 가능해야만 한다면 이것이 필요하다. TCP 또는 UDP처럼 transport level에 있는 transport protocols와는 대조적으로, "wire protocol"이란 용어는 application level에서 정보를 표현하기 위한 일반적 방법을 설명하는데 사용된다.

2) the transport layer or **layer 4**
이것에서는 네트워크 구성요소들과 프로토콜들의 layed architecture에 있는 어플용을 위해 end-to-end communication services를 제공한다. The transport layer는 connection-oriented data stream support, reliability, flow control, and multiplexing과 같은 편리한 서비스를 제공한다. Transport layers는 인터넷의 기초인 TCP/IP model (RFC 1122) 그리고 일반적인 네트워킹에 대한 the Open Systems Interconnection (OSI) model, 둘 다에 포함되어 있다.

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 3세대로 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

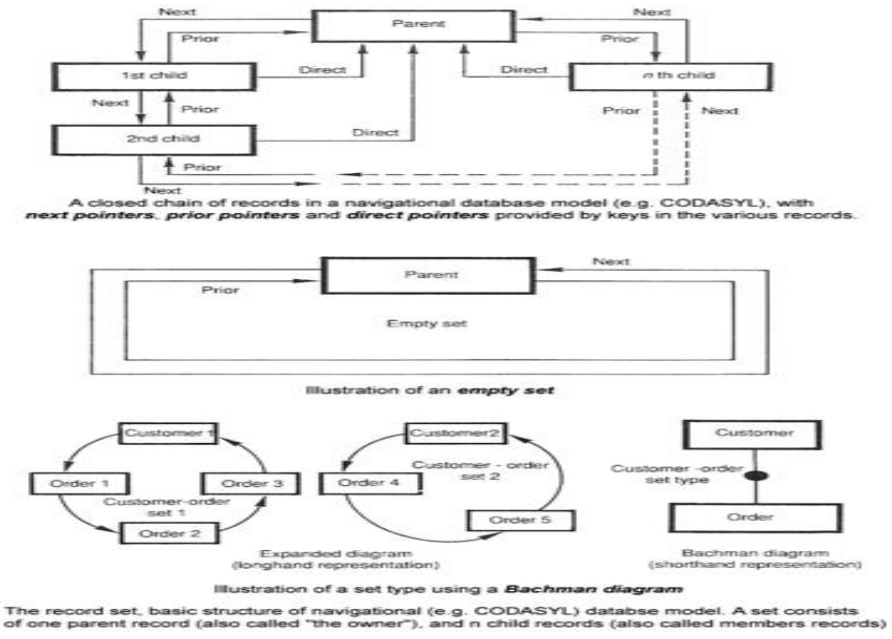
다른 사물로부터 나온 레퍼런스를 우선적으로 추적하여 레코드와 사물을 찾을 수 있는 데이터베이스의 한 종류이다. Navigational interfaces은 대체로 procedural이지만, XPath와 같은 몇 가지 현대적인 시스템들은 에서는 동시에 navigational and declarative하다고 여겨질 수 있다.

Navigational access은 전통적으로 데이터베이스 인터페이스의 the network model and hierarchical model과 결합되며, 어떤 것은 set-oriented features을 가지고 있다. 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".

이것은 많은 컴퓨터에서 사용할 수 있는 표준 프로그래밍 언어의 개발을 유도하기 위하여, 1959년에 생겨난 consortium이다. 이들의 노력이 COBOL과 기타 표준들의 개발을 이끌었다.

<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**

database는 관계형 데이터베이스에서 사용된 tabular 관계보다는 다른 수단으로 모델화된 데이터의 저장 및 검색 메카니즘을 제공한다. 이러한 시도의 동기는 simplicity of design, horizontal scaling and finer control over availability 이다. NoSQL databases는 가끔 간단한 검색과 추가 기능을 기본적으로 고려함으로써 key-value 저장을 높게 최적화할 수 있다. 반면에 RDBMS는 범용의 데이터 저장용으로 여겨지고 있다.

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.

4. 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.

1) in-memory 데이터베이스

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 있었다. 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.

2) active 데이터베이스

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.

3) cloud 데이터베이스

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.

4) data warehouse

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.

5) deductive 데이터베이스

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.

6) distributed 데이터베이스

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.

7) document-oriented 데이터베이스

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.

8) embedded 데이터베이스 시스템

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.

9) end-user 데이터베이스

는 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.

10) federated 데이터베이스 시스템

은 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.

11) multi-database

이 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.

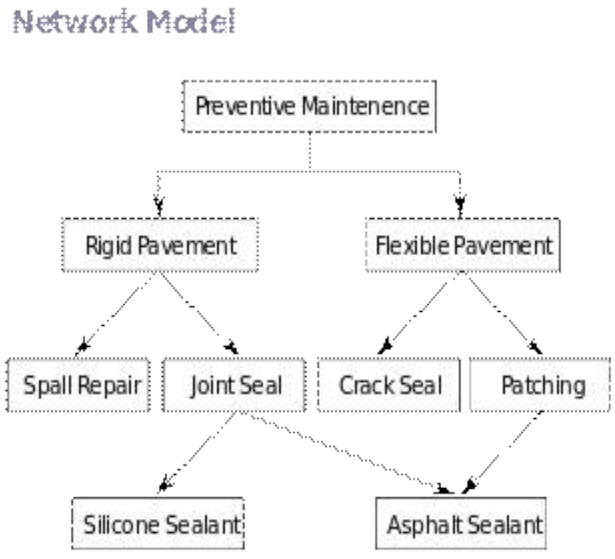
(3) A **distributed transaction** is an operations bundle, in which two or more network hosts are involved. Usually, hosts provide transactional resources, while the transaction manager is responsible for creating and managing a global transaction that encompasses all operations against such resources.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

12) graph 데이터베이스

NoSQL 데이터베이스의 일종이며, 정보를 저장하고 표현하기 위하여 nodes, edges, properties 를 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.



13) hypertext 또는 hypermedia 데이터베이스

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.

14) knowledge base(KB, kb)

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.

15) 모바일 데이터베이스

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.

16) operational 데이터베이스

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.

17) parallel 데이터베이스

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.

18) Real-time databases

곧바로 실행되거나 회수되어야 하는 결과를 충분하고도 신속하게 교신을 통해 실시간으로 처리한다.

19) Spatial database

다차원적인 모습으로 데이터를 저장할 수 있다. 그런 데이터에 대한 쿼리에는 “이 지역에서 가장 가까운 호텔은?”과 같은 위치기반 쿼리가 포함된다.

20) 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.

21) Terminology-oriented database

객체지향형 데이터베이스에서 만들어지며, 종종 특정한 분야로 제한된다.

22) 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.

5. 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 한 가지 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 정확하게 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.

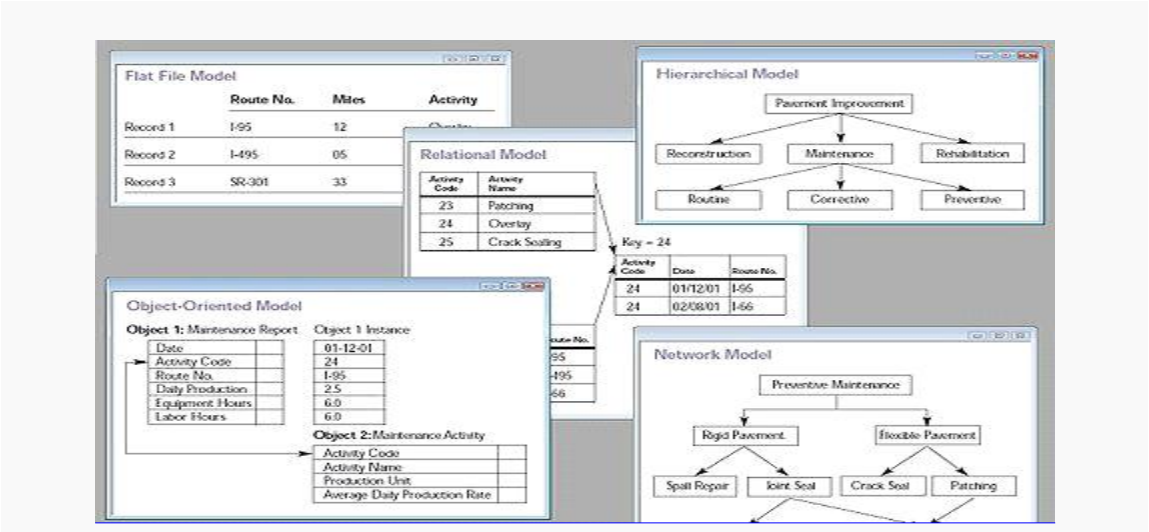
1) **Database normalization**
s the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate 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.

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.

5.1. Database models



Collage of five types of database models.

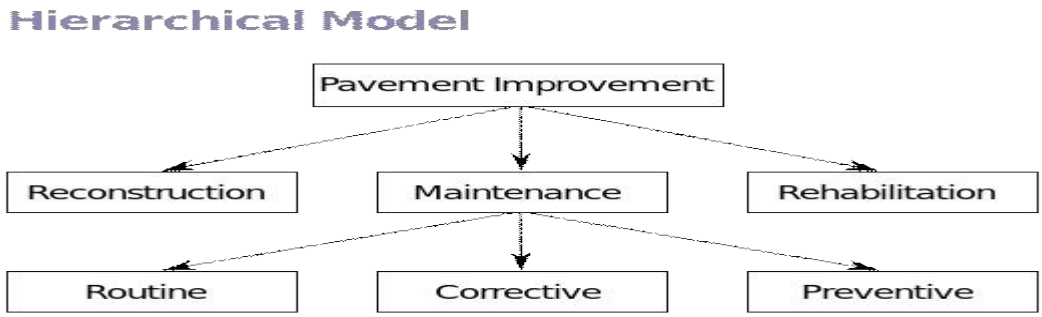
데이터베이스 모델이란 기본적으로 어떠한 방식으로 데이터를 저장, 조직, 취급할 것인지를 다루는 데이터베이스의 논리적 구조를 결정하며 데이터 모델의 한 유형이다. 가장 인기 있는 데이터베이스 모델은 테이블을 근거로 하는 형식의 관계형 모델(또는 관계형과 유사한 SQL 모

델) 이다.

데이터베이스용의 일반적인 논리적 데이터 모델에는 다음과 같다:

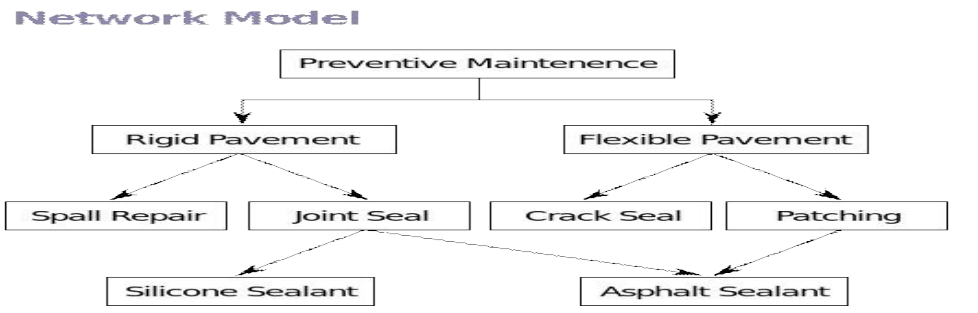
(1) Hierarchical database model

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.

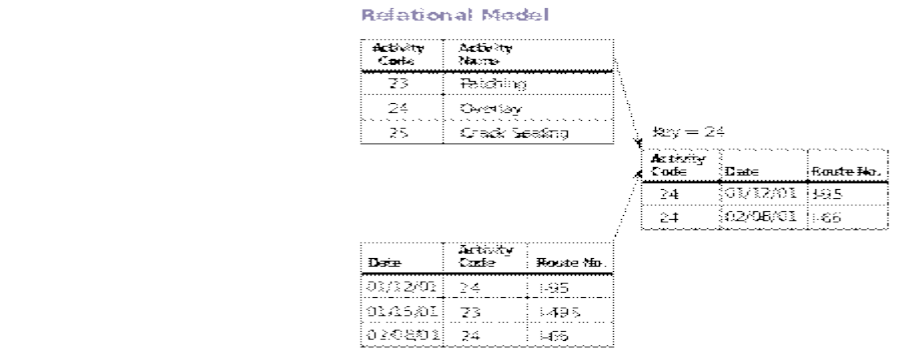
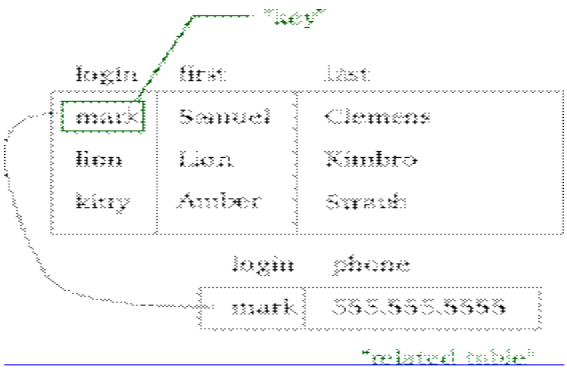


Diagram of an example database according to the Relational model.



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.

(5) Semantic model

씨멘틱 데이터 모델은 소프트웨어 공학에서 여러 가지의 의미를 가지고 있다:

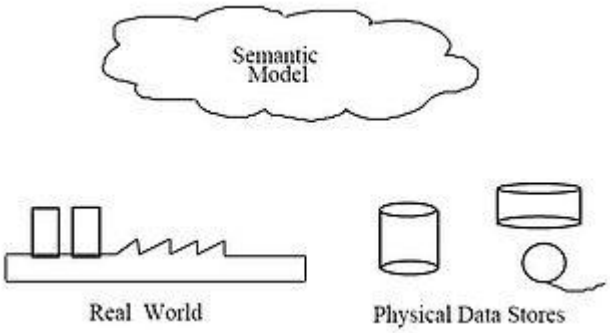
1. 어의 정보를 포함하고 있는 개념적 데이터 모델인데, 이 뜻은 그것의 instances에 대한 의미를 기술하는 모델이란 것이다. 이러한 어의적 데이터 모델은 저장된 심볼(the instance data)이 실세계와 어떻게 관련되어 있는지를 설명하는 하나의 abstraction이다.

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.

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."



(6) 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.

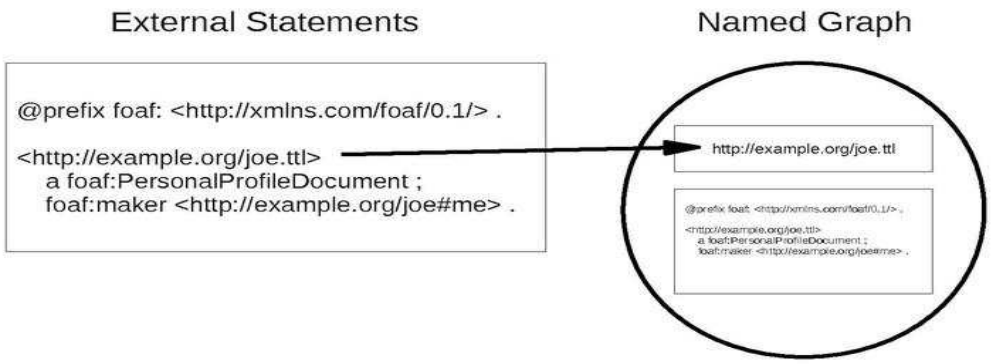
1) In computing, a **persistent data structure** is a data structure that always preserves the previous version of itself when it is modified. Such data structures are effectively immutable, as their operations do not (visibly) update the structure in-place, but instead always yield a new updated structure. (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.")

(7) 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>

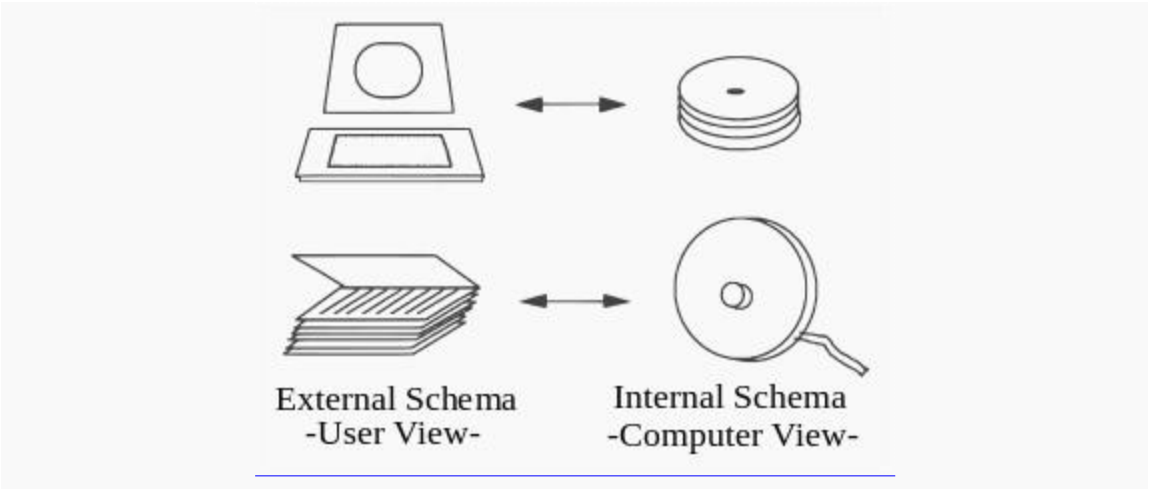
웹의 한 가지 개념화는 URIs로 식별된, 그리고 HTML 다큐멘트에서 표현된 hyperlink arc로 연결된 다큐멘트 nodes의 그래프이다. (대체로 웹 브라우저를 통해) URI에서 HTTP GET를 실행함으로써, 어느 정도(somehow)-연관된 다큐멘트가 검색될 수 있다. 이러한 "follow your nose" 어프로치 또한 Linked Data(전형적으로 RDF syntax가 statements 시리즈처럼 데이터를 표현하는데 사용되는 형태)와 다른 자원으로 가기 위하여 RDF point 속에 있는 URIs의 형태로 웹에 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).

5.2 DBMS의 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 독립

성과 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 데이터메이스를 지배하고 있는 관계형 데이터베이스 모델의 실행에서 중요한 특징들이다.

6. 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

6.1 SQL이란?

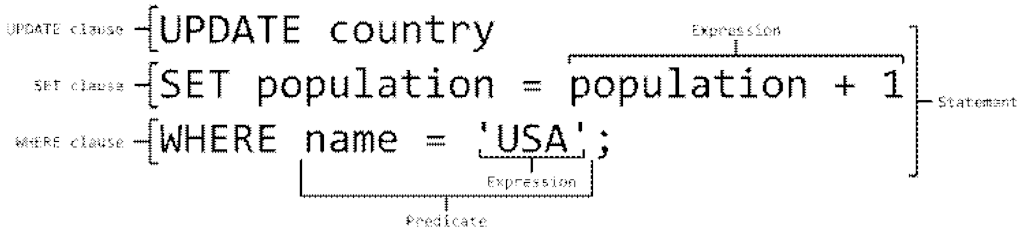
SQL(Structured Query Language)이란 relational database management system (RDBMS)에 포함된 데이터를 관리하도록 디자인된 또는 relational data stream management system (RDSMS)에서 stream processing를 위한 특별한 목적의 프로그래밍 언어이다.

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.

6.2 Syntax



6.3 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 whitespace
가독성을 위하여 SQL 코드를 쉽게 포맷할 수 있도록, SQL statements와 queries에서 무시된다.

6.4 Operators

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

6.5 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 소스 안에서 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 디폴트 된다. 그렇지만 표준의 “simple case”와 달리, Oracle's DECODE는 서로 평등하게 두 개의 NULLs를 고려하고 있다.

6.6 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 return 하야 한다는 것을 특정화하기 위하여 사용될 수 있다. SELECT는 SQL에서 그것에 포함되는 선택적 키워드와 절(clauses)과 더불어 가장 복장한 statement이다:

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

►WHERE 절
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 절
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 절
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 절
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;
```

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 벤더들이 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;
```

6.7 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.

6.8 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, Unknown 이란 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.

True와 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.

6.9 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 ...])
```

6.10 Transaction controls

Transactions은 필요하면 DML 연산자들을 포장(wrap)할 수 있다.

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.

COMMIT와 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;
```

6.11 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
우 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;
?
?
?
```

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

1) Character strings

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

2) Bit strings

► BIT(n): an array(배열, 배열된 데이터 군) of n bits
► BIT VARYING(n): an array of up to n bits

3) Numbers

► INTEGER, SMALLINT and BIGINT
► FLOAT, REAL and DOUBLE PRECISION
►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

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

► 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.

6.13 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;
```

6.14 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 장점

을 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 빼) 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.

6.16 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.

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.

7.1 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.

7.1.1 Free the database of modification anomalies

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 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.

7.1.2 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.

7.1.3 Make the data model more informative to users
정규화된 테이블, 그리고 하나의 정규화된 테이블과 다른 것과의 관계는 실세계의 개념과 그것간의 상호연관성을 반영한다(mirror).

7.1.4 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.

7.2 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.

7.3 Background to normalization: definitions

7.3.1 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.

7.3.2 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.

7.3.3 Transitive dependency(전의적 의존성)

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

7.3.4 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.

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

7.3.6 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.

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

7.3.4 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).

7.3.5 Non-prime attribute(비-으뜸 속성)
비-으뜸 속성이란 어떠한 후보키에서도 생기지 않는 속성이다. Employee Address는 "Employees' Skills" 테이블에 있는 비-으뜸 속성이다.

7.3.6 Prime attribute(으뜸 속성)
으뜸 속성은 역으로 어떤 후보키에서 생기는 속성이다.

7.3.7 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.

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

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)

7.4.1 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.

7.4.1.1 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.

7.4.1.3 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 string: 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 ISBN: 원자적이지 않다고 여겨질 수 있는데, 그것에는 언어와 출판사 식별자가 포함되어 있기 때문이다.

Date가 제안하길, "the notion of atomicity *has no absolute meaning*": a value란 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.

7.4.1.4 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 따라

서 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.

7.4.2 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 2NF이라는 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.

7.4.2.1 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 한번이

상(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 수

행되지 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.

7.4.2.2 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

7.4.3 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.

7.4.3.1 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.

7.4.3.2 "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 생년월인을 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.

7.4.3.3 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.

7.4.3.4 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.

7.4.4 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 기본키이거나 그것의 일부이다.

7.4.4.1 Example

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

7.4.5 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) $X \rightarrow Y$ is a trivial functional dependency ($Y \subseteq X$)
2) X is a superkey for schema R

7.4.5.1 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 속성들도 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.

7.4.5.2 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 설명한 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. 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.

8.1 Database storage

데이터베이스 기억장치(storage)란 데이터베이스의 물리적 형체(materialization)의 container다. 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.

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.

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

8.1.1 Database materialized views

가끔 메모리의 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 메로리의 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.

8.1.2 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.

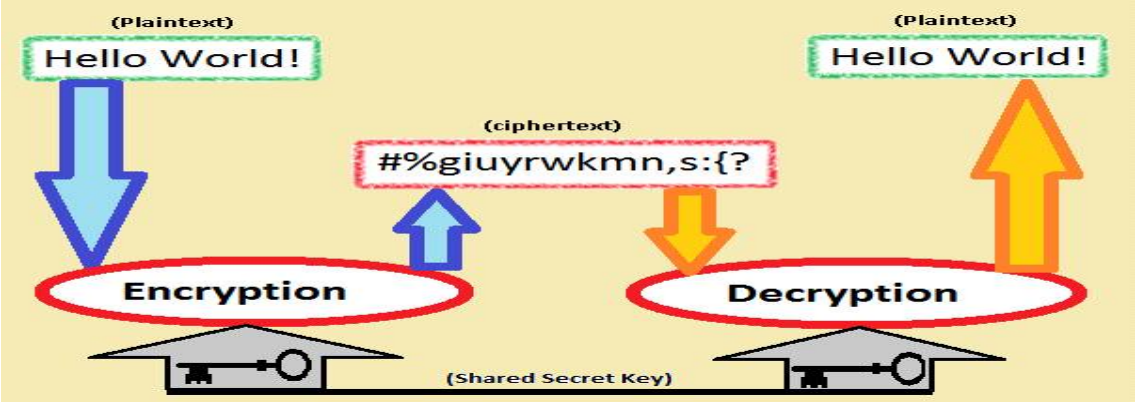
8.2 Database security

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.



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

8.3 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 설명하는 두문자어 이다: 원자가, 일관성, 독립성, 인내성(Atomicity, Consistency, Isolation, and Durability).

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

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

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

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

8.4 Migration

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.

8.5 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 의

해 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

8.6 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.

8.7 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.

9. Linked Data And The Semantic Web

<http://www.linkeddatatools.com/semantic-web-basics>

Linked Data와 Semantic Web이란 무엇인가? 원칙적으로 Semantic Web은 Web 3.0 이다; 즉, 시스템들 또는 엔티티(entities) 끼리 서로 데이터를 링크하는 방법이다. 따라서 웹 환경에서는 이것을 사용하여 데이터끼리 풍부하고, 자아-기술적(self-describing)인 관계성(interrelations)을 갖는다.

사실, SW는 HTML 다큐멘트에 포함되어 있는 데이터를 사람이 읽는 것이 아니라, 컴퓨터가 읽을 수 있어야 한다는 사고의 전환에서 비롯되었다. 다시 말해서, 컴퓨터가 인간을 위해 좀 더 많은 사고적(thinking) 업무를 수행할 수 있어야 한다는 생각에서 출발하였다.

1) Semantic search란?
Seeks to improve search accuracy by understanding the searcher's intent and the contextual meaning of terms as they appear in the searchable dataspace, whether on the Web or within a closed system, to generate more relevant results. Semantic search systems consider various points including context of search, location, intent, variation of words, synonyms, generalized and specialized queries, concept matching and natural language queries to provide relevant search results.[1] Major web search engines like Google and Bing incorporate some elements of semantic search.

Guha et al. distinguish two major forms of search: navigational and research. In navigational search, the user is using the search engine as a navigation tool to navigate to a particular intended 다큐멘트. Semantic search is not applicable to navigational searches. In research search, the user provides the search engine with a phrase which is intended to denote an object about which the user is trying to gather/research information. There is no particular 다큐멘트 which the user knows about and is trying to get to. Rather, the user is trying to locate a number of 다큐멘트s which together will provide the desired information. Semantic search lends itself well with this approach that is closely related with **exploratory search**.

How Does It Differ From The Web As It Is Today?

오늘날 우리가 웹에서 얻는 많은 데이터는 웹 페이지의 형태로 우리에게 전달되는데, 이것들은 HTML 다큐멘트들이며, 서로 하이퍼링크로 연결되어 있다. 사람이나 기계 모두 이 같은 다큐멘트를 읽을 수 있다. 그렇지만, 사람은 전형적으로 페이지에서 키워드를 찾을 수 있지만, 기계가 이 다큐멘트에 들어 있는 의미를 스스로 발췌하기란 결코 쉽지 않다.

Enter Linked Data - Liberating Web Databases From Their Old Chains

웹에는 많은 정보가 들어 있지만, 전형적으로 raw data 그 자체를 이용할 수는 없다. 만일 데이터베이스의 웹 사이트가 만들려 한다면, 그것의 HTML 다큐멘트들은 해당 데이터로만 만들어져야 한다.

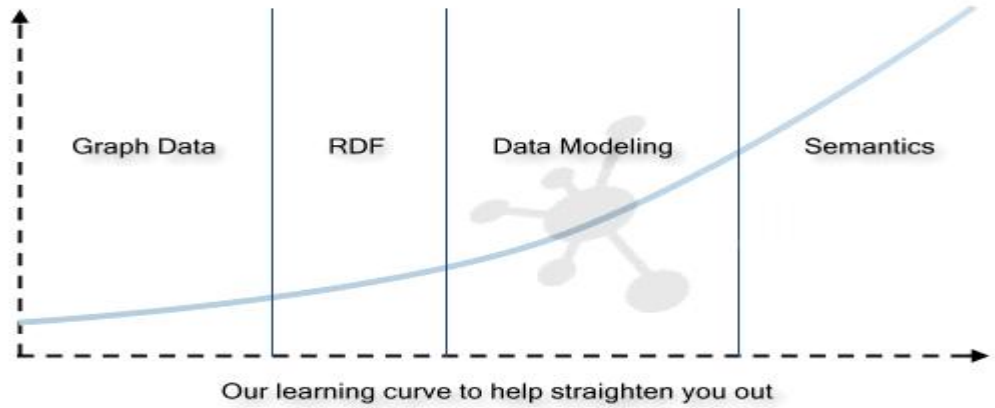
따라서 SW은 이 같은 문제를 해결하기 위하여, 다음과 같은 여러 가지 방법으로 현재의 인터넷 모습을 변화시키고 있다:

- 인공지능처리를 위하여 web of data를 개방하고 있다.(웹으로 하여금 우리를 위해 약간 좀 생각하도록 한다.)
- 회사, 조직, 개인들로 하여금 공개된 표준 포맷으로 자유롭게 자신들의 데이터를 출판하도록 한다.
- 기업에서 이미 웹에 있는 데이터를 이용할 수 있도록 한다(데이터 주고받기)

실재로, SW는 여러 곳에서 출판된 모든 HTML 정보를 수집한 다음, 그것을 마치 전에는 하나의 데이터베이스였던 것처럼, 다루고 조사하는 데이터 모델의 한 유형이다.

오늘날의 다양한 관련 도구와 소프트웨어를 비교해 볼 때, 이것이 인터넷을 사용하는 모든 data humanity 분야의 자동화 연구에 끼치는 이익은 엄청나다.

But Where Do I Start?



이 강의는 복잡하지 않다. 우리는 초보자의 시작을 돕기 위하여 개론으로 시작한 다음에, SW를 정의하고, SW과 현재의 웹과의 차이를 알아보며, 마지막으로 SW의 제작 기술에 대하여 깊이 있게 알아볼 것이다:

- ►Tutorial 1 Introduction To Graph Databases - gives a brief overview of the way in which the semantic web stores data.

- ►Tutorial 2 RDF - A Quick Start - an introductory look at Resource Description Framework (RDF), the format the semantic web uses to store data in graph databases.

► Tutorial 3 Semantic Modeling - introduces the key aspects of describing data with meaning, or semantics - and the tremendous advantages this can offer.

► Tutorial 4 Introduction To RDFS & OWL - the key syntax the semantic web uses to encode semantic meaning into data.

► Tutorial 5 Querying Semantic Data - how to query published semantic data using SPARQL protocol - the means to harness the immense discovery capabilities of the semantic web.

Tutorial 1: Introducing Graph Data

SW은 그렇게 잘 알려진 분야가 아니다. 초보자가 SW의 정의와 그것의 작동원리를 열심히 이해하려 한다면, 먼저 그것의 데이터 저장방법부터 이해해야 한다. 따라서 먼저, SW의 데이터 저장 모델인 graph database부터 알아보다.

After this tutorial, you should be able to:

- Describe in basic terms what the semantic web is.
- Experience the paradigm-shift of storing information as a graph database, rather than a hierarchical or relational database.
- Understand that the semantic web of data is defined using Resource Description Framework (RDF).
- Understand the basic principles of RDF statements and how they can define data graphs.

여러분이 전통적인 IT 분야에 지식을 갖고 있고, 계층형(for example XML) 또는 관계형 데이터베이스(for example MySQL, MS SQL)의 데이터저장방법에 대하여 알고 있겠지만, Resource Description Framework(RDF)에 대해선 잘 알지 못할 수도 있다.

RDF란 SW 커뮤니티에서 사용하는 일반적인 두문자어이며, 시멘틱 데이터의 웹을 제작하는데 필요한 기본적인 빌딩블록들 중의 한 요소이다. 또한 이것이 define(정의)하는 것은 여러분이 익숙하지 않은 데이터베이스의 유형인 **graph database** 이다.

2) define이란?
- To state the precise meaning of (a word or sense of a word, for example).

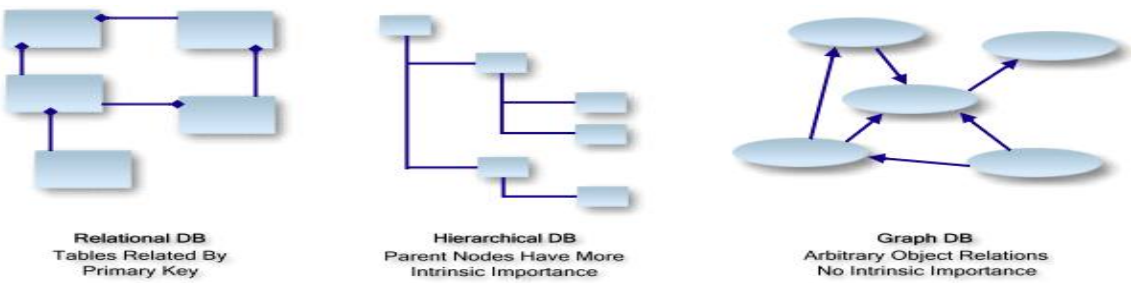
- To describe the nature or basic qualities of; explain:

비록 이 데이터베이스에 대해 여러분이 생소하더라도, 이것이 바로 전 세계적으로 SW을 제작하는데 사용하는 데이터베이스의 한 부류 이다.

이제 graph database가 무엇이고, 어떻게 RDF가 그것을 정의하는지, 그리고 graph database를 시각화하는 방법이 무엇인지에 대하여 알아보기로 한다.

먼저 hierarchical, relational, 그리고 graph databases를 비교하여 이것들이 서로 어떻게 다른지를 살펴보기로 하자:

1.1 Introducing The Graph Database



대부분의 데이터 저장 유형들에서는, 자신들의 여러 가지 데이터 요소(elements) 중에는 다른 것보다 보다 더 많은 precedence나 importance를 갖고 있는 몇 가지의 요소들(예를 들어, data nodes 또는 data tables)에 대한 개념을 정의하고 있다.

예를 들어, XML 다큐멘트를 보자. 전형적으로 XML 다큐멘트에는 한 개의 parent node와 함께 여러 개의 정보 노드를 사용하고 있다. 따라서 이러한 다큐멘트의 root는 최상위의 노드이며, 이 노드는 어떠한 부모 노드도 갖지 않는다.

위의 그림을 살펴보자. 맨 오른쪽의 data graph에는 어떠한 roots(또는 계층)도 존재하지 않으며, 서로 연결된 자원들로만 구성되어 있다. 또한 이 그래프의 어떠한 자원도 특별하게 다른 자원보다 본질적으로(intrinsic) 더 중요하지 않다는 것을 나타내고 있다.

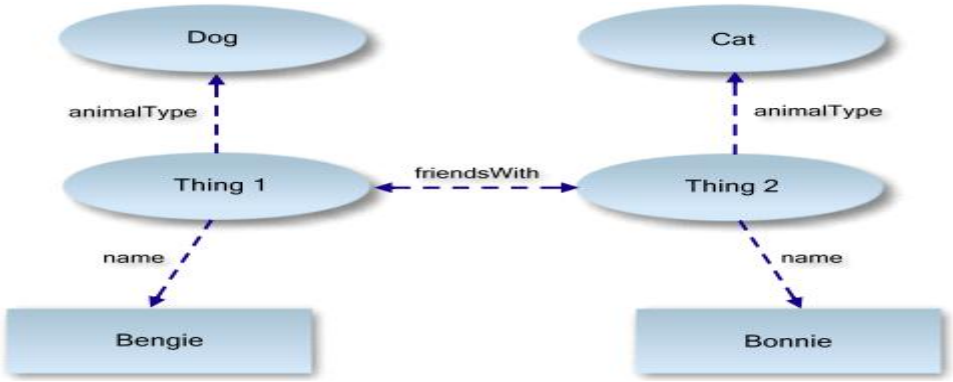
An Example Of A Data Graph

사물들(things)이 서로 어떤 관계를 갖고 있는지를 기술하고 있는 지시문(statements)과 그 속에 있는 관계들을 RDF로 표현하는 방법을 알아보기에 앞서서, 먼저 그래프로 이들 사물간의 관계를 시각화해 보자. 이것은 매우 쉽다.

먼저 아래의 개(벤지)와 고양이(보니) 간의 관계를 설명하는 지시문을 살펴보자:

■ Bengie는 개다.
■ Bonnie는 양이다.
■ Bengie와 Bonnie는 친구이다.

위의 간단한 3개의 지시문을 데이터 그래프로 그려보면, 아래와 같다:



위의 그래프에 나타나 있는 관계는 매우 직관적이므로, 우리는 이 관계들을 완전하게 이해할 수 있다: 즉, 우리는 “Thing 1" 과 "Thing 2" 라는 두 개의 사물이 있고, 이 사물들의 속성 이름이 하나는 ”animalType“이고 나머지는 ”friendsWith“라는 것을 알 수 있다.

또한, 우리는 "Thing 1"'의 속성 “name”의 값이 “Bengie”이고, "Thing 2"의 속성 “name”의 값이 “Bonnie”라는 것도 알 수 있고, 물론 "Thing 1"은 개를, "Thing 2"는 고양이를 의미한다는 것도 알 수 있다. 끝으로, 이 두 개의 사물은 서로 서로 친구(속성 “friendsWith”가 화살표로 양 방향 모두를 가리키고 있음)라는 것도 알 수 있다.

핵심 포인트 - 위의 그래프에서 화살표는 속성(properties)을 나타낸다: 때론, 이것을 RDF 용어(terminology)로는 predicates라 한다. 이 두 용어는 호환적으로 사용되며, 그래프에서 표시된 화살표는 속성을 의미한다.

<Graph Model의 예: 지시문>

http://dbpedia.org/resource/Billie_Jean has a **singer** whose value is **Michael Jackson**

► Subject: http://dbpedia.org/resource/Billie_Jean (URI)
► Predicate: http://www.example.com/terms/singer (URI)
► Object: Michael_Jackson (Literal)

<URI란?>
정보기술에서, Uniform Resource Identifier (URI)는 자원을 식별하기 위하여 사용되는 문자열(a string of characters) 이다. 이러한 식별 기능을 통하여 WWW과 같은 네트워크에 있는 자원의 표현물(representations) 간의

상호작용이 이루어질 수 있다. 가장 잘 알려진 URI의 일반적 형태는 종종 비공식적으로 웹 어드레스로 불리우는 Uniform Resource Locator (URL) 이다. 또한 거의 사용되지는 않고 있는 Uniform Resource Name (URN)이 있는데, 이것은 특별한 namespaces에 있는 자원들을 식별하기 위한 메카니즘을 제공함으로써 URLs을 보완하도록 디자인된 것이다.

예를 들어, Uniform Resource Name (URN)을 사람의 이름이라면, Uniform Resource Locator (URL)은 그들이 사는 거리의 어드레스라 비유할 수 있다. 다시 말해서, URN은 아이템을 식별하기 위한 것이고, URL은 그것을 찾는 방법을 제공하는 것이다.

공식적으로 RDF에 대해 알아보기 전에, 다음과 같은 간단한 예를 먼저 맛보기로 하자:

1.2 A Starting Example Of RDF

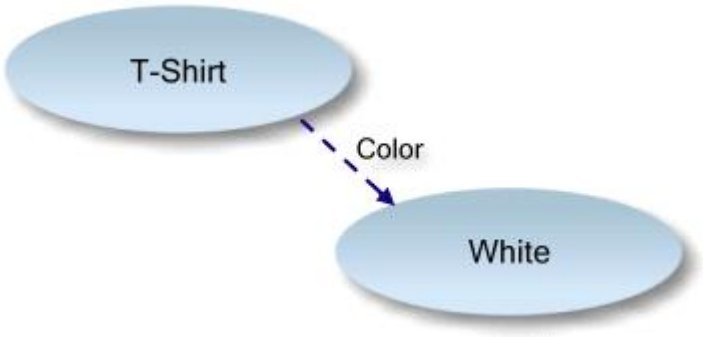
```
01. <?xml version="1.0" encoding="UTF-8"?>
02.
03. <rdf:RDF
04.       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
05.       xmlns:dc="http://purl.org/dc/elements/1.1/"
06.       xmlns:region="http://www.country-regions.fake/">
07.
08.     <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Oxford">
09.           <dc:title>Oxford</dc:title>
10.           <dc:coverage>Oxfordshire</dc:coverage>
11.           <dc:publisher>Wikipedia</dc:publisher>
12.           <region:population>10000</region:population>
13.           <region:principaltown rdf:resource="http://www.country-regions.fake/oxford"/>
14.     </rdf:Description>
15.
16. </rdf:RDF>
```

지금 위의 내용에 대해 걱정하지 마라. 우리는 나중에 이것을 다시 검토할 것이다. 지금은 단지 위와 같은 것이 RDF/XML - the XML form of RDF라는 것만을 이해하면 된다. RDF를 레코딩하는 여러 가지 방법들이 있지만, 우리는 단지 RDF/XML만을 살펴볼 것이다.

1.3 The RDF Statement (Triple)

위의 콘텐트에서 붉은 색으로 표시한 RDF/XML(<rdf:Description> tags 사이)를 RDF 지시문, 또는 때때로 RDF triple이라 부른다. 이 두 가지 이름 중에서 triple이 우리가 RDF를 이해하는데 가장 도움이 되는 용어이데, 그 이유는 이것이 지시문을 3가지의 요소로 지시문이 구성되어 있다는 것을 의미하기 때문이다: 즉, 지시문의 subject, predicate, 그리고 object.

그래프의 형태로 이 3가지의 구성 용어들을 나타내 보자. 예를 들어, 티-셔츠의 색깔을 표현하고 있는 데이터에 대한 다음의 그래프를 살펴보자:



위의 간단한 그래프에서는 3개의 구성요소가 표현되어 있다:

- Subject: T-shirt;
- Predicate(또는 property): color;
- Object: white.

핵심 포인트 - RDF는 SW의 데이터 구조를 정의하는 토대이지만, 데이터에 숨어있는 어의나 의미를 스스로 기술하지는 못한다. 이것은 나중에 RDFS (RDF Schema) 와 OWL (Web Ontology Language)을 배울 때 다루기로 한다. 지금은 이것들에 대하여 걱정하지 마라. 먼저, RDF가 데이터간의 관계구축방법이 이미 잘 알려져 있는 기존의 데이터저장방법과 어떻게 다른지에 대해 배워야 한다. 또한 여러분은 계층형이나 관계형 데이터 모델에서 벗어나 graph model로 추세가 바뀌고 있음을 깨닫는 것이 중요하다.

간단한 RDF/XML 지시문을 통해, 이 구성요소들에 대하여 알아보자:

```
01 .<?xml version="1.0" encoding="UTF-8"?>
02.
03. <rdf:RDF
04.       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
05.       xmlns:feature="http://www.linkeddatatools.com/clothing-features#">
06.
07.       <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
08.
09.             <feature:color rdf:resource="http://www.linkeddatatools.com/colors#white"/>
10.
11.       </rdf:Description>
12.
13. </rdf:RDF>
```

공식적으로 우리는 이 지시문을 다음 강의에서 분석하겠지만, 여러분은 먼저 위의 RDF에서 subject, predicate, object를 어떻게 기술하는지에 대하여 감을 잡아야 한다.

이제 이장을 마감한다. 이제 여러분은 다음과 같은 것을 이해하고 있을 것이다:

- What a data graph is.
- That the semantic web is a giant, global data graph defined in RDF (Resource Description Framework).
- The all-important shift in thinking from storing data in relational, or hierarchical models to a storing in graph models.
- The subject, predicate and object in terms of basic data graphs and RDF statements.
- A basic familiarity with the layout of an RDF document.

Tutorial 2: Introducing RDF/XML

graph data model이 SW에서 데이터를 저장하는 모델이라면, RDF는 그것을 만드는 포맷이다. 앞 장에서, 우리는 graph data와 RDF를 살펴봤다. 이제 우리는 RDF/XML - 웹에서 가장 인기 있는 RDF 포맷들 중의 하나 - 을 사용하여 그래프 데이터를 작성하는데 필요한 기본 개념을 설명할 것이다.

이미 여러분은 graph database의 개념을 살펴봤고, 계층형과 관계형 데이터 모델과 같은 전통적인 데이터 저장 형태와 이것을 비교도 해 보았다.

이제 간단하게 RDF (Resource format)을 살펴보고, 주체(subject), 술어(predicate 또는 property), 그리고 객체(object)로 이루어진 지시문을 어떻게 정의하는지에 대해서도 알아봤으며, subject->predicate->object relationship를 triple이라 부른다는 것도 알았다. 또한 여러분은 RDF가 시멘틱 데이터의 웹을 구축하는 기본 포맷이라는 것도 배웠다.

이번 강좌에서, 여러분은 한 단계씩 여러분 자신의 RDF 지시문을 구축하는 방법을 배울 것이고, 그래프를 사용하여 그것들을 시각적으로 이해할 수 있게 될 것이다. 그런 다음에, 데이터에 어의나 의미를 추가하는 것에 대해 생각해 볼 것이고, 끝으로 이것이 제공하는 커다란 장점에 대해서도 이해하게 될 것이다.

위의 내용을 가장 잘 수행할 수 있도록, 먼저 간단한 RDF 다큐멘트의 예를 가지고 한 단계씩 알아보기로 한다.

2.1 Building An RDF document

>Add The RDF document Root Tag

먼저, RDF root node를 다음과 같이 추가해 보자:

```
1. <rdf:RDF
2.       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
3.
4.        <!-- Body Code Omitted -->
5.
6. </rdf:RDF>
```

위의 예에 있는 두 번째 줄에서, 여러분은 표준화된 W3.org namespace인

http://www.w3.org/1999/02/22-rdf-syntax-ns#

라는 URI를 보게 될 것이다. 이 namespace는 다른 컴퓨터 리더(reader)에게 이 태그로 봉해져 있는(enclosing) 다큐멘트가 RDF 다큐멘트라는 것과 여기서 사용된 rdf:RDF tag들이 이 namespace에 포함되어 있다는 것을 알려주는 것이다.

1) 컴퓨팅에서 namespace란 여러 종류의 사물을 조직하기 위하여 사용되는 심볼의 세트를 말한다. 따라서 이러한 사물들은 이름으로 참조(referred to)될 수 있다: 잘 알려진 예로는 다음과 같은 것이 있다:

- file systems은 파일들에 이름을 할당한 namespaces 이다;
- programming languages는 namespaces에 있는 자신들의 변수와 하위-루틴을 조직한다;
- 컴퓨터 네트워크와 분산 시스템은 computers, printers, websites, (remote) files, 등과 같은 자원에 이름을 할당한다.

Namespaces는 일반적으로 다른 환경에서도 그 이름을 재사용할 수 있도록 계층구조를 갖는다. 하나의 추론으로, 각각의 사람은 올바른 이름뿐만 아니라 자신들의 친척과 공유하고 있는 가족이름을 갖고 있는 인명 시스템을 생각해 보자. 만일, 각 가족에서 가족 이름의 고유한 것이라면, 각각의 사람은 이름과 가족이름의 결합에 의해 유일하게 식별될 수 있다. 다시 말해서, 비록 많은 Janes이 있더라도 Jane Doe는 단지 한명 뿐이다; Doe라는 성의 namespaces에서, 단지 "Jane"은 이 사람을 확실하게 설명하는데 충분하지만, 모든 사람의 "global" namespace에서는 full name이 사용되어야만 한다.

비슷한 방법으로, 계층적 파일 시스템은 디렉토리로 파일을 조직한다. 각 디렉토리는 하나의 독립된 namespace이다. 그러므로 디렉토리 "letters" 와 "invoices"는 둘 다 "to_jane" 파일을 포함할 수도 있다.

컴퓨터 프로그래밍에서, namespaces는 전형적으로 특별한 기능성과 관련된 symbols 과 identifiers를 그룹핑할 목적으로, 그리고 동일한 이름을 공유하는 복수의 identifiers 간의 충돌을 피할 목적으로 사용된다.

네트워킹에서, **Domain Name System**은 websites와 기타 자원들을 namespaces로 조직화 한다. 예를 들어, "org"는 비영리기관용 namespace이다. 예를 들어, "wikipedia.org"는 Wikimedia Foundation에 할당된 하위 namespace이며, "en.wikipedia.org"는 이 스페이스에 있는 영어판 위키피디어의 이름이다.

위의 다큐멘트에서 namespace를 표현하고 있는 RDF node가 바로 이 RDF 다큐멘트의 roots node이다.

>Add A Statement

RDF 다큐멘트에는 하나 이상의 지시문이 포함될 수 있다. 간단하게 우리는 하나를 추가할 것이다. RDF/XML에서 주체(subject)를 정의하는 방법은 <rdf:Description> tag를 사용하는 것이다. 다음과 같이 붉은 색 지시문을 추가해 보자:

```
01. <rdf:RDF
02.     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
03.
04.     <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
05.
06.         <!-- Statement Code Omitted -->
07.
08.     </rdf:Description>
09.
10. </rdf:RDF>
```

위에서 붉은 색으로 표현된 <rdf:Description> tag가 의미하는 것은 간단하게 말해서 다음과 같다:

"나는 객체 t-shirt에 대하여(about) 정의(describe)하며, 그것의 유일한 ID는 http://www.linkeddatatools.com/clothes#t-shirt 이다."

주목!!!
<rdf:Description>은 about 속성에 의해 지정된 resource에 대한 URI 정보를 갖고 있는 container 이다. 다음의 예에서 각각의 resources는 books이고, 그것의 식별자(URI)는 서명이다. 그리고 각각의 책에는 한명의 저자와 페이지 수만 표현되어 있다.

Source

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:lib="http://www.zvon.org/library">

  <rdf:Description about="Matilda">
    <lib:creator>Roald Dahl</lib:creator>
    <lib:pages>240</lib:pages>
  </rdf:Description>
```

!!! 위이 <rdf:Description />의 또 다른 표기법:

```
    <rdf:Description about="Matilda" lib:creator = "Roald Dahl" lib:pages="240" />

    <rdf:Description about="The BFG">
      <lib:creator>Roald Dahl</lib:creator>
      <lib:pages>208</lib:pages>
    </rdf:Description>

    <rdf:Description about="Heart of Darkness">
      <lib:creator>Joseph Conrad</lib:creator>
      <lib:pages>110</lib:pages>
    </rdf:Description>

    <rdf:Description about="Lord Jim">
      <lib:creator>Joseph Conrad</lib:creator>
      <lib:pages>314</lib:pages>
    </rdf:Description>

    <rdf:Description about="The Secret Agent">
      <lib:creator>Joseph Conrad</lib:creator>
      <lib:pages>249</lib:pages>
    </rdf:Description>
</rdf:RDF>
```

Output

Author Title Pages
Roald Dahl
Matilda
240
Roald Dahl
The BFG
208
Joseph Conrad
Heart of Darkness
110
Joseph Conrad
Lord Jim
314
Joseph Conrad
The Secret Agent
249

좀 이해가 됐나요? 계속 배워봅시다.

>Add Predicates

여러분이 주체에 대해 정의하면서 그것의 고유한 ID를 지정해 놓았지만, 그 주체의 특성에 대

해서는 어떠한 것도 정의하지 않았다. RDF 지시문에서는 RDF 전문용어로 속성을 의미하는 properties나 predicates를 사용하여 해당 주체의 특성들을 정의한다.

간단하게, T-shirt의 속성 중 하나인 size를 다음과 같이 추가해 보자:

```
01. <rdf:RDF
02.     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.     xmlns:feature="http://www.linkeddatatools.com/clothing-features#">
04.
05.     <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
06.
07.         <feature:size>12</feature:size>
08.
09.     </rdf:Description>
10.
11. </rdf:RDF>
```

위의 7번째 줄을 보자. 간단하게 말해서 이 주체는 <feature:size> 태그이름으로 기술된 한 개의 속성을 갖고 있으며, 이 속성의 문자 값은 12이다. RDF 전문용어로 이것이 바로 지시문(statement)이다.

그리고 3번째 줄에 이 속성이름뿐만 아니라 이 객체에서 사용되는 속성이름의 namespace에 대한 URI가 표시되어 있다는 것을 확인하라.

끝으로, T-shirt의 색깔인 color 속성을 하나 더 다음과 같이 추가해 보자:

```
01. <rdf:RDF
02.     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.     xmlns:feature="http://www.linkeddatatools.com/clothing-features#">
04.
05.     <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
06.
07.         <feature:size>12</feature:size>
08.         <feature:color rdf:resource="http://www.linkeddatatools.com/colors#white"/>
09.
10.     </rdf:Description>
11.
12. </rdf:RDF>
```

이 <feature:color> 속성의 표현은 <feature:size>와 다르다는 것을 알았을 것이다. 지난 번 속성에서는 문자 값 12가 있었던 반면에, 이번 것에는 또 다른 지시문의 subject(ID)를 참조하도록 지정해 놓았다. 이러한 표현이 옳은 것이다. RDF에서 object는 다른 지시문에 있는 subject를 참조할 수 있다.

!!! 속성 rdf:resource는 다른 rdf:Description element에서 정의하고 있는 resource에 about한 정보를 입력하는데 사용될 수 있다.

Book Sample

```
<rdf:Description about="RD">
<lib:firstName>Roald</lib:firstName>
<lib:surname>Dahl</lib:surname>
</rdf:Description>

<rdf:Description about="JC">
<lib:firstName>Joseph</lib:firstName>
<lib:surname>Conrad</lib:surname>
</rdf:Description>

<rdf:Description about="Matilda">
<lib:creator rdf:resource='RD'/>
<lib:pages>240</lib:pages>
</rdf:Description>

<rdf:Description about="The BFG">
<lib:creator rdf:resource='RD'/>
<lib:pages>208</lib:pages>
</rdf:Description>

<rdf:Description about="Heart of Darkness">
<lib:creator rdf:resource='JC'/>
<lib:pages>110</lib:pages>
</rdf:Description>

<rdf:Description about="Lord Jim">
<lib:creator rdf:resource='JC'/>
<lib:pages>314</lib:pages>
</rdf:Description>
```

다시 본론으로 돌아가서, <feature:color>의 속성은 3번째 줄에 있는 xmlns:feature에 이미 포함되어 있다는 것도 이해하여야 한다. 다시 말해서, xmlns:feature의 이름리스트에는 이미 size와 color라는 속성이름이 포함되어 있다는 것이다.

!!! RDF 다큐멘트에 있는 주체 또한 다른 RDF 지시문에서 속성의 객체처럼 참조될 수 있다. 이것은 RDF 초보자들에게는 개념적으로 혼란스러울 수 있다.

따라서 이것은 간단하게 "이 주체는 ID가 http://www.linkeddatatools.com/colors#white인 지시문을 참조하고 있는 객체와 더불어 feature:color이라는 이름의 한 개의 속성을 가지고 있다"라고 말하고 있다.

2.2 Breaking Down The Statement

이제 간단한 예의 RDF 다큐멘트를 살펴보고, 우리가 배운 것을 근거로 지시문의 구성부분을 분석해 보자:

```
1.  <rdf:Description rdf:about="subject">
2.        <predicate rdf:resource="object" />
3.        <predicate>literal value</predicate>
4.  <rdf:Description>
```

이미 여러분은 지시문의 주체(what the statement is about), 그리고 두 가지 형태의 속성(다른 RDF 지시문을 참고하도록 하는 resources 그리고 문자값과 에 대하여 알게 되었다.

주목 - rdf:Description element는 단일 container 안에 하나 이상의 지시문을 집단화할 수 있게 허용하고 있다. 위와 같은 일반적인 형태에는 사실상 한 개의 속성과 한 개의 객체 즉, literal과 resource, 두 가지가 함께 동일한 주체를 참조하는 두 개의 지시문이 포함되어 있다.

2.3 A More Thorough Example

앞에서 그래프 데이터에서 다루었던 보다 복잡한 예로 되돌아가 보자:

```
01.<?xml version="1.0" encoding="UTF-8"?>
02.
03.<rdf:RDF
04.xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
05.xmlns:dc="http://purl.org/dc/elements/1.1/"
06.xmlns:region="http://www.country-regions.fake/">
07.
08.<rdf:Description rdf:about="http://en.wikipedia.org/wiki/Oxford">
09.<dc:title>Oxford</dc:title>
10.<dc:coverage>Oxfordshire</dc:coverage>
11.<dc:publisher>Wikipedia</dc:publisher>
12.<region:population>10000</region:population>
13.<region:principaltown rdf:resource="http://www.country-regions.fake/oxford"/>
14.</rdf:Description>
15.
16.</rdf:RDF>
```

여러분의 이해력을 테스트하고 보완할 분야를 알기 위하여, 위의 RDF 다큐멘트를 아래와 같이 구분할 수 있는지를 스스로 확인해 봐라:

- Subject of the statement?
- Predicates of the statement(including whether they are resources or literals)?

■ Objects referenced by the **resource** predicates?

이제 RDF 다큐멘트s에 대해 이해했고, 그것이 데이터 그래프와 어떤 관계가 있는지를 알았다면, 여러분은 RDF graph data로 시멘틱즈를 모델링하는 방법에 대하여 알 준비가 끝난 것이다.

2.4 A Quick Recap Of URIs And XML Namespaces

앞에서 사용된 http://www.linkeddatatools.com/clothes#t-shirt처럼 우리가 이제까지 사용해 왔던 고유한 IDs를 Uniform Resource Identifiers, 또는 줄여서 URIs라 부른다. 우리는 이제까지 아무런 설명없이 지시문의 주체, 속성, 객체에 고유한 IDs를 제공하기 위하여 URIs를 사용해 왔다.

URIs는 전 세계적으로 데이터 교환을 가능하게 만들자는 RDF의 목적을 달성하는데 너무나 중요하기 때문에, 우리는 이제 신속하게 URIs에 대해 다시 살펴보자.

>XML Namespace URIs

앞에 있던 RDF 다큐멘트 예로 다시 가 보자. T-shirt size 속성은 아래의 7번째 줄에서 <feature:size>란 이름을 가지고 있다는 것을 알 수 있다:

```
01.<rdf:RDF
02.xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.xmlns:feature="http://www.linkeddatatools.com/clothing-features#">
04.
05.<rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
06.
07.<feature:size>12</feature:size>
08.<feature:color rdf:resource="http://www.linkeddatatools.com/colors#white"/>
09.
10.</rdf:Description>
11.
12.</rdf:RDF>
```

그리고 3번째 줄에서, 여러분은 우리가 XML namespace feature를 정의하기 위하여 그것에 URI http://www.linkeddatatools.com/clothing-features#라는 namespace를 지정하였다는 것도 주목해야 한다.

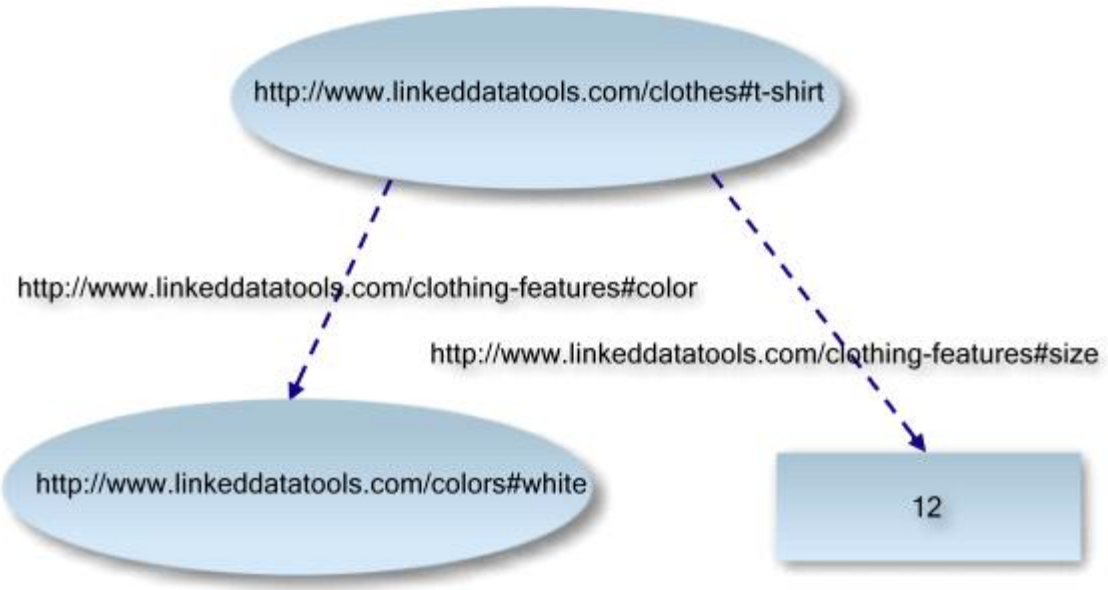
이 namespace의 목적은 단지 동일한 이름의 태그로 인하여 발생하는 이름간의 충돌을 피하기 위한 것이다: 그렇지만 “size”란 이름의 태그가 다른 namespace URIs로 정의된다면, RDF

reader는 비록 그것들이 동일한 태그이름라 하더라도 서로 다른 속성들이라는 것을 알게 된다.

또한 추가로 feature:size용으로 완전한 자격을 갖춘 URI를 얻기 위하여, 간단하게 prefix feature를 그것의 완전 이름인 namespace URI인 http://www.linkeddatatools.com/clothing-features#size로 대체할 수 있다.

Note - RDF에서 XML namespace URIs는 동일한 (태그) 이름을 가진 속성들을 구분하는데 사용된다. 충분히 자격이 있는 URI를 얻기 위하여, 간단하게 namespace prefix를 그것의 namespace URI로 대체할 수 있다.

이제 우리는 완전하게 자격을 갖춘 URIs에 대해 말할 수 있다:



보시다시피, 이것을 RDF graph diagrams으로 완전하게 URIs를 표현하는 것이 항상 쉽지도 편리하지도 않다. 종종 간략 버전(shorthand versions)이 대신 사용된다(다시 말해서, namespace prefix만을 사용하는 것).

이 번 강의를 마치자. 여러분은 다음의 방법에 대해 이해하여야 한다:

- How to write your own basic RDF 다큐멘트s in RDF/XML
- Understand how and why RDF uses URIs to identify subjects, predicates and objects
- How to relate an RDF 다큐멘트 to a corresponding data graph with fully qualified URIs

Tutorial 3: Semantic Modeling

RDF가 전 세계적으로 교환 가능한 데이터를 레코딩하기 위하여 유연하고 그래프-중심의 모델을 제공하는 반면에, 그것이 어떤 **어의나 의미**를 제공하지는 않는다. 이제 일반적으로 이용 가능한 데이터 모델을 검토해서 문제점(fuss)이 무엇인지에 대해 알아보자.

데이터를 모델화하는 많은 인기 있고 주류적인 방법들이 있다. 이것들 중에서 어떤 것들은 다른 것보다 최신식이다. RDF 모델의 장점을 조사하기 전에, 이미 기존의 데이터 모델 방법들 중에서 몇 가지를 살펴보기로 하자:

아래의 테이블에서 시멘틱 데이터 모델의 고유한 특성에 초점을 맞추어, 몇 가지의 방법들을 비교해 보았다.

3.1 Comparing The Popular Data Models

데이터 모델링의 주류 방법과 SW 모델 간의 특징(features) 비교

Model	Example Format	Data	Metadata	Identifier	Query Syntax	Sematics (Meaning)
Object Serialization	.NET CLR Object Serialization	Object Property Values	Object Property Names	e.g. Filename	LINQ	N/A
Relational	MS SQL, Oracle, MySQL	Table Cell Values	Table Column Definitions	Primary Key (Data Column) Value	SQL	N/A
Hierachical	XML	Tag/Attribute Values	XSD/DTD	e.g. Unique Attribute Key Value	XPath	N/A
Graph	RDF/XML, Tutle	RDF	RDFS/OWL	URI	SPARQL	Yes, using RDFS and OWl

***serialization?**
In the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment.

***XSD?**
XSD (XML Schema Definition), a recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language (XML) 다큐멘트.

***DTD?**
A 다큐멘트 type definition (DTD) is a set of markup declarations that define a 다큐멘트 type for an SGML-family markup language (SGML, XML, HTML). A 다큐멘트 Type Definition (DTD) defines the legal

building blocks of an XML 다큐멘트. It defines the 다큐멘트 structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML 다큐멘트, or as an external reference.

*XPath?
XPath, the XML Path Language, is a query language for selecting nodes from an XML 다큐멘트. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML 다큐멘트. XPath was defined by the World Wide Web Consortium (W3C).

3.2 Why Include Semantics In Data? Knowledge Integration

만일 시멘틱이 커다란 이익을 주지 못한다면, 우리는 우리의 데이터에 시멘틱을 추가할 필요가 없다. 데이터에 시멘틱 의미를 추가하는 가장 중요한 이익들 중의 하나는 자동적으로 다양한 지식의 영역에 가지를 뻗칠 수 있다는 것이다. 무엇이 지식의 도메인(domains of knowledge)인가? 간단한 예를 살펴보자.

위의 예에서, 두 개의 웹 사이트들은 서로 독립적으로 시작되었다. 한 사이트는 현재와 과거의 Oscar 수상 영화에 대한 정보를 호스트하고 있으며, 다른 사이트는 할리우드 남녀 영화배우의 전기에 대한 대규모의 데이터베이스 이다.

그렇지만 둘 다 자신들의 웹 사이트 데이터베이스에 보완할 정보가 있다. 우리는 먼저 이들 사이트 간에 시멘틱의 사용 없이 어떻게 정보공유가 발생하는지를 살펴보고 난 다음에, 시멘틱즈를 사용하여 동일한 정보를 두 사이트가 서로 공유하는 방법에 대하여 알아볼 것이다.

1) Sharing Without Semantic Modeling

두 사이트 중에서 하나는 모든 오스카 수상영화에 대한 MS SQL database이고, 또 하나는 할리우드 배우에 대한 MySQL database이다. 이것들은 http://www.oscarwinners.fake 그리고 http://www.actorbiographies2go.fake에 각각 포함되어 있다. 그리고 이 두 사이트는 독립적으로 시작하였고 협력하지도 않고 있다.

Oscar Winners site는 그 이름처럼 과거에 제작된 모든 오스카 수상영화를 리스트하고 있으며, 또한 그것들에서 주연을 맡은 남녀 영화배우의 리스트도 갖고 있다. 그렇지만, 그 콘텐트에

는 이들의 이름과 생일날짜 이외의 다른 정보는 소장하고 있지 않다.

Actor Biographies 사이트는 현재뿐만 아니라 과거의 많은 할리우드 배우들에 대하여 완전한 전기 리스트뿐만 아니라 그들이 출연한 영화 리스트도 가지고 있다. 그러나 이것의 콘텐트에는 해당 영화에 대한 어떠한 film-plots(영화구성), 또는 screen-shot(image)을 가지고 있지 않다.

이제 이 두 사이트가 최신의 모델보다 기존의 전통적 데이터 모델에서 협력할 수 있는 방법에 대하여 살펴보자:

■분명하게 말해서, http://www.oscarwinners.fake의 이용자는 출연배우의 이름을 클릭하면, 그들에 대해 더 많은 정보를 찾을 수 있다는 이점이 있다. - 이 정보는 http://www.actorbiographies2go.fake의 MySQL database에 저장되어 있다.

■똑같이, http://www.actorbiographies2go.fake의 이용자가 출연배우들의 영화이름을 클릭하여 더 많은 정보를 얻을 수 있다. 이것은 http://www.oscarwinners.fake에 있는 MS SQL database에 저장되어 있다.

■두 사이트간의 어떠한 데이터 공유도 자신들의 데이터베어스에 있는 테이블을 결합(joining)시키지 못하고 있다. 먼저, 이것들은 처음부터 독립적으로 설계되었으며, 따라서 두 데이터베이스에 있는 각각의 영화배우나 영화를 참조하는 primary key를 동기화(synchronized)시킬 수 없다. 이 문제를 해결하기 위하여 이것들은 mapped되어야 할 것이다. 그러나 또하나의 문제는 이것들이 서로 호환성이 없는 데이터베이스 서버 시스템을 사용하고 있다는 것이다.

■자신들이 현재 사용하고 있는 데이터베이스 간의 협력을 위하여, 각 사이트의 소유자들은 공동으로 영화 및 배우에 대한 고유한 ID scheme을 만들어 정보를 공유할 수 있는 공동의 데이터 포맷을 결정해야 할 것이다. 예를 들어, 자신들의 웹사이트에서 서로의 필요에 따라 정보를 요구할 수 있는 안전한 XML 콘텐트를 만든다면, 이것이 가능할 것이다. 이런 방식으로 자신들의 정보 공유를 발전시킬 수 있다.

Important Point - 비호환적이고 독립적인 데이터 시스템 간에 정보를 교환하는 것은 시간, 돈, 그리고 서로 다른 데이터 세트에 대한 상황판단에 의한 인간적 해석을 필요로 한다. 또한 이러한 두 개의 웹사이트의 데이터 도메인으로만 제한되어 있어서 다른 곳에서 지식을 추가하는 데도 비슷한 노력이 요구된다. 따라서 이것은 인간이 데이터의 의미를 이해하도록 두 데이터베이스를 올바르게 통합시키는 공동 포맷을 필요로 한다.

이제 RDF와 semantics을 소개하기로 한다. RDF와 semantic web을 사용하여 이러한 문제를 해결할 수 있는 방법에 대하여 조사해 보자 - 모든 것은 자동적으로 이루어지며, 수동적으로 이루어지진 않는다.

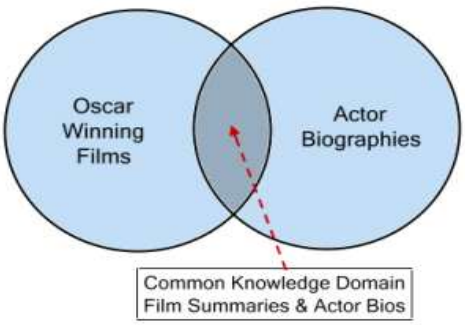
2) Sharing With The Semantic Web Model

시멘틱 모델링에서, 먼저 다음과 같은 중요한 용어에 대해 이해하여야 한다:

■ Vocabulary – contexts 간에 정의가 잘 일치하는 용어들의 집합이다.
■ Ontology – 우리로 하여금 정의된 vocabulary 뒤에 숨어있는 contextual relationships를 정의한다. 이것은 지식의 영역을 정의하는 cornerstone이다. 온톨로지를 정의하기 위한 공식적 syntax는 OWL (Web Ontology Language)이며, 이것은 RDFS (RDF Schema)를 확장시켜 놓은 것으로, 다음 레슨에서 설명하기로 한다.

***온톨로지란?**
정보시스템의 대상이 되는 분야에 존재하는 개체와 개념에 대한 명세로서, 사람과 컴퓨터간에 공유되는 지식을 개념화한 구체적인 형식이며, 개념화와 개념화간의 관계를 표현하는 것이다.온톨로지는 단어와 관계들로 구성된 일종의 사전으로서 생각할 수 있으며, 그 속에는 특정 도메인에 관련된 단어들이 계층적으로 표현되어 있고, 추가적으로 이를 확장할 수 있는 연관관계가 포함되어 있어, 웹 기반의 지식 처리나 응용프로그램 사이의 지식 공유 등이 가능하도록 되어 있다. 즉, 시맨틱 웹의 목적인 자동적인 실행과 추론을 하기 위해 온톨로지는 가장 핵심적인 개념이라고 할 수 있다.

시멘틱 모델링을 사용하여 우리가 두 개의 사이트의 시나리오를 어떻게 모델 하는가?



첫째, 두 사이트들은 공동의 표준적인 vocabulary를 개발하여, 이것을 문맥상에서 (contextually) 일치하는 자신들의 데이터를 기술하는 적용시켜야 한다. 예를 들어, 용어 'film title'은 두 사이트 모두에서 동일한 사물(thing)을 의미하여야 하며, 용어 'actor name' 과 'actor birthdate'도 마찬가지 이다.

이러한 활동으로 vocabulary로 사용된 용어들을 가지고 데이터의 암시적 의미를 표현할 수 있으며, 또한 쿼리에 의한 동일한 데이터를 생산할 수 있다. 이러한 결과는 똑같은 base ontology, 또는 common vocabulary를 채택하고 있는 두 사이트에 의해 얻어질 수 있다. 결과적으로 이들 두 사이트는 웹에서 서로 통신(communicate)할 수 있다.

만일 이러한 표준 vocabulary가 적재적소에 사용된다면(in place):

■ 두 사이트는 동일한 용어에 의해 서로 질의할 수 있다.

■ The Oscar Winning Movies site는 on-demand 방식으로 the Actor Biographies site의 배우 이름을 쿼리할 수 있으며, 그 영화에 출연한 특별한 남녀배우에 대하여 보다 자세한 정보를 얻을 수 있다.
■ The Actor Biographies site 역시 on-demand 방식으로 Oscar Winning Movies site에 있는 film plots을 이제 쿼리할 수 있으며, 배우가 출연했던 영화들에 대하여 더 많은 자세한 정보를 얻을 수 있다.
■ 공식적인 웹 온톨로지에서 정의된 contextual relationships(문맥전후 연관성)와 더불어, 촬영장소와 같은 영화배우나 영화에 대한 추가적 연관 정보, 영화촬영과 동일한 날짜에 발생한 여러 가지 뉴스, 배우의 생년월일, 또는 동일한 제작자에 의해 제작된 영화 등등, 처음부터 그러한 정보가 존재했다고 상상하지 못하더라도, 이용자는 이러한 링크용 표준용어(linked standard terminology)를 사용하여 부수적 정보를 얻을 수도 있다.
■ 이것은 두 사이트간에 존재하는 transformation, mapping, 또는 contracts와 같은 필요성과 상관없이 이루어져야 한다. 모든 정보가 바로 이 의미론(semantics)을 통해 생산된다.

우리는 다음 강의에서 SW 온톨로지의 구성(makeup)이 무엇이고, 여러분이 시멘틱 데이터베이스에 어떻게 쿼리하는지, 그리고 심지어 그것에서 기계적 추론(machine inference)을 어떻게 형성하는지를 보여줄 것이다.

Point Of Interest - 좋은 뉴스는 여러분이 관심을 갖는 특별한 지식영역에 대하여 여러분 스스로 온톨로지를 정의하고 공유하려는 노력을 기울일 필요가 없다는 것이다. 웹에는 이미 여러분이 채택할 수 있는 수 많은 인기있는 표준 온톨로지가 존재하고 있다. 그리고 필요하다면 여러분 스스로 이것을 확대할 수도 있다. 다음 섹션에서 이들 중 몇 가지를 소개하고자 한다.

여기서 논의된 도메인간의 지식공유(cross-domain knowledge sharing)는 단지 웹 사이트에만 적용되는 것이 아니라, 기관에서 구축한 지식 base에서도 적용된다. 따라서 SW 테크놀로지는 웹에서 출판된 어플이나 정보에만 제한되지 않아야 한다.

비록 시멘틱 데이터베이스를 처음 구축할 때, 좀 더 많은 기초 작업이 필요하다 하더라도, 전 세계에 걸쳐서 도메인 간의 통합의 용이성, 절약된 시간, 그렇게 하여 얻어진 아이디어에 대한 이점들은 잠재적으로 매우 귀중한 것이다.

3.3 Metadata Initiatives

지식의 도메인에서 용어를 표현하는 표준 vocabularies, 또는 공식적 ontologies는 이미 다양한 주제 – 예를 들어, media terms, biomedical terms, scientific terms – 에 대한 표준 vocabularies를 만드는 여러 전문기관으로부터 무료로 이용할 수 있다. 몇 가지 예를 살펴보면 다음과 같다:

■ Dublin Core Metadata Initiative (DCMI) - 특히 공통적이고 일상적인 용어 그리고 미디어의 주요 용어에 특별하게 초점을 맞추면서 여러 주제에 대한 온톨로지를 만들고 있다.

■ Friend Of A Friend (FOAF) - social networking purposes에 맞는 표준 vocabulary/ontology의 개발에 초점을 맞추고 있다.
■ OpenCyc(오픈사이크) - 일상적이고 상식적인 지식을 수집해 놓은 온톨로지.
The OpenCyc Platform is your gateway to the full power of Cyc, the world's largest and most complete general knowledge base and commonsense reasoning engine. OpenCyc contains hundreds of thousands of Cyc terms organized in a carefully designed ontology.

이번 강의는 끝났다. 다음 강의에서, 우리는 SW에서 온톨로지를 정의하는 방법과 SW 데이터베이스에서 정보를 쿼리하는 방법에 대한 보다 완벽한 기술적 내용을 알아볼 것이다.

You should now understand the following:

■ The main benefits of semantic data over traditional data models.
■ How a semantic web application might function to automatically link data between independent data sources covering the same base of knowledge.
■ That open standards for common, everyday vocabulary currently exist.

Tutorial 4: Introducing RDFS & OWL

앞장의 데이터 모델에서 vocabulary와 semantics의 모델링에 따른 장점을 소개하였고, 이제 의미론과 함께 RDF data models을 다루는데 필요한 실질적인 기술을 소개하기로 한다. RDF data는 RDFS 그리고 OWL과 같은 두 가지의 syntaxes를 사용하여 시멘틱 메타데이터와 함께 encoded 될 수 있다:

지난 강의에서, 우리는 시멘틱 모델과 함께 데이터를 모델화하는 보다 인기있고 전통적인 형태의 몇 가지를 비교하였다. 그리고 또한 SW 방식을 사용함으로써, 데이터 공유를 보다 분명하게 그리고 쉽게 확대시키고 상황(situation)을 소개하였다.

그리고 샘플을 사용하여, 두 개의 독립된 웹사이트가 서로 데이터를 공유함으로써 이익을 얻는 상황도 보여주었다. 이번 강의에서, 우리는 semantic metadata와 함께 RDF 데이터를 표현(annotate)하는데 사용되는 공식적인 syntax를 조사할 것이다. 그리고 그 다음 강의에서, 우리는 이런 데이터를 출판하고 쿼리하는 방법을 알아볼 것이다.

RDF 데이터는 두 가지의 중요한 신텍스인 RDFS와 OWL을 사용하여 시멘틱 메타데이터와 함께 표현된다. RDFS 와 OWL 둘 다 W3C specifications 이다.

4.1 A Starting Example

여러분이 보고 있는 것이 OWL의 예이다:

```
01. <rdf:RDF
02. xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03. xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
04. xmlns:owl="http://www.w3.org/2002/07/owl#"
05. xmlns:dc="http://purl.org/dc/elements/1.1/">
06.
07. <!-- OWL Header Example -->
08. <owl:Ontology rdf:about="http://www.linkeddatatools.com/plants">
09. <dc:title>The LinkedDataTools.com Example Plant Ontology</dc:title>
10. <dc:description>An example ontology</dc:description>
11. </owl:Ontology>
12.
13. <!-- OWL Class Definition Example -->
14. <owl:Class rdf:about="http://www.linkeddatatools.com/plants#planttype">
15. < rdfs:label>The plant type</rdfs:label>
16. <rdfs:comment>The class of plant types.</rdfs:comment>
17. </owl:Class>
18.
19. </rdf:RDF>
```

지금 자세히 알려고 하지마라. 나중에 살펴보기로 한다. 그렇지만, 주목해야 하는 것은 예제인 RDF 다큐멘트의 header에 전에는 없었던 두 개의 새로운 namespaces가 포함되어 있다는 것이다: 3번째 줄의 RDFS (RDF Schema, http://www.w3.org/2000/01/rdf-schema#) 그리고 4번째 줄의 OWL (Web Ontology Language, http://www.w3.org/2002/07/owl#)의 namespaces 이다.

한 가지 더 주목할 것은 RDF에서 우리의 온톨로지를 정의하는 방법이다. 그래, 온톨로지 또한 RDF 다큐멘트 이다.

전통적인 온톨로지 문서를 분해해 보자. 예로서, plant varieties(식물의 종류)를 정의하고 있는 간단한 온톨로지를 살펴보자.

Point Of Interest - Why OWL, not WOL? When the acronym for Web Ontology Language (OWL) was first proposed by the working group, OWL was adopted instead of WOL as it is easily remembered, and suggested wisdom. But confusingly enough, it is still an acronym that should strictly speaking be WOL.

4.2 OWL Header

```
01. <rdf:RDF
02.  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
04.  xmlns:owl="http://www.w3.org/2002/07/owl#"
05.  xmlns:dc="http://purl.org/dc/elements/1.1/">
06.
07. <!-- OWL Header Example -->
08. <owl:Ontology rdf:about="http://www.linkeddatatools.com/plants">
09. <dc:title>The LinkedDataTools.com Example Plant Ontology</dc:title>
10. <dc:description>An example ontology written for the LinkedDataTools.com RDFS &OWL introduction
    tutorial</dc:description>
11. </owl:Ontology>
12.
13.<!-- Remainder Of 다큐멘트 Omitted For Brevity... -->
14.
15.</rdf:RDF>
```

비록 온톨로지가 헤더에 포함되지 않아야 하더라도, 이 곳은 여러분의 온톨로지에 포함되어 있는 것을 다른 사람에게 이해하는데 도움이 될 정보를 포함시키기에 좋은 장소이다.

위에서처럼, 우리는 온톨로지용 한 개의 title과 한 개의 description을 포함시켰다. 그렇지만, 이곳은 또한 올바른 갱신을 나타내는 version information을 포함시켜야 하는 장소이며, 여러분의 온톨로지를 다른 온톨로지에 수출(import)한다고 진술(state)할 수 있는 장소이다.

만일 여러분의 온톨로지가 다른 온톨로지에서 온 elements를 사용한다면, 이것은 여러분의 온톨로지가 다른 것에 의존하고 있다는 것을 알리는 tools나 frameworks를 위해 절대적으로 필요한 것이다.

!!! Point:
dc: prefix,http://purl.org/dc/elements/1.1/라는 namespace를 정의한 5번째 줄을 봐라. 이것은 Dublin Core Metadata Initiative라는 namespace를 참조하며, machine readers에게 **dc:title** and **dc:description** 와 같은 엘리멘트들은 이 온톨로지에서 정의하고 있다고 알려주고 있다. Remember - we mentioned this often used ontology at the end of the previous tutorial. And, because it's an OWL ontology, it is also defined in RDF. Just point your browser to its namespace URI to download it.

4.3 OWL Classes, Subclasses & Individuals

온톨로지의 제 1차적 목적은 things를 semantics나 meaning에 따라 분류하는 것이다. OWL에서, 이런 목적은 this is achieved through the use of *classes* 그리고 *subclasses*를 사용하여 이룰 수 있다. 예를 들어 OWL에서는 이것을 *individuals(instance)* 라 부른다. 특정한 OWL class의 멤버들인 individuals는 그것의 *class extension*라 부른다.

OWL에서 *class* 는 individuals를 분류하여 공통의 성질을 공유하고 있는 그룹이다. 만일 하

나의 individual이 한 class의 한 멤버라면, 그것은 machine reader에게 그것이 OWL class에 의해 주어진 어의적 분류에 속한다고 알려준다.

An Example

다시 OWL ontology의 예를 보자. 이번에는 몇 가지의 classes와 subclasses가 추가 되었다. 우리는 3가지의 plant classes를 정의한다: the **flowering plants** class, **shrubs** class, 그리고 이 두가지 클라스를 섭클라스로 가지고 있는 **planttype** class. 따라서 **planttype** class는 모든 plant types의 최상위 클라스이다.

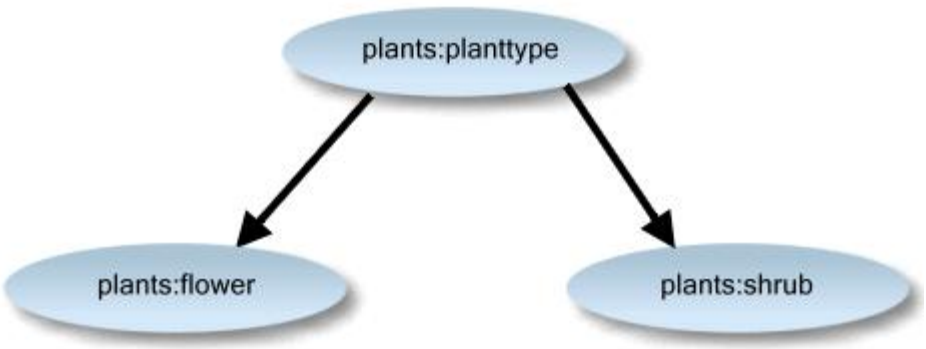
```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:plants="http://www.linkeddatatools.com/plants#">

<!-- OWL Header Omitted For Brevity -->

<!-- OWL Class Definition - Plant Type -->
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#planttype">

<rdfs:label>The plant type</rdfs:label>
<rdfs:comment>The class of all plant types.</rdfs:comment>

</owl:Class>

<!-- OWL Subclass Definition - Flower -->
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#flowers">

<!-- Flowers is a subclassification of planttype -->
<rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#planttype"/>

<rdfs:label>Flowering plants</rdfs:label>
<rdfs:comment>Flowering plants, also known as angiosperms.</rdfs:comment>

</owl:Class>

<!-- OWL Subclass Definition - Shrub -->
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#shrubs">

<!-- Shrubs is a subclassification of planttype -->
<rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#planttype"/>

<rdfs:label>Shrubbery</rdfs:label>
<rdfs:comment>Shrubs, a type of plant which branches from the base.</rdfs:comment>
```

```
37.
38. </owl:Class>
39.
40. <!-- Individual (Instance) Example RDF Statement -->
41. <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">
42.
43. <!-- Magnolia is a type (instance) of the flowers classification -->
44. <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
45.
46. </rdf:Description>
47.
48. </rdf:RDF>
```

Point Of Interest(RDF 다큐멘트 의 타당성 조사) - You can investigate this RDF 다큐멘트 further by passing it through W3C's RDF validator, which automatically tabulates the 다큐멘트's RDF subjects, predicates and objects for you. This can help significantly improve your understanding of RDF. Just click the 'copy to clipboard' button on the top right of the code excerpt and paste into the validator.

Taxonomy - A Hierarchy Of Terms

우리가 한 것은 우리의 어의적 용어나 클래스를 계층적으로 정의한 것이다. SW 세계에서, 이러한 용어의 계층을 *taxonomy*라 부른다. 다음은 우리가 정의한 taxonomy hierarchy을 그래프로 나타낸 것이다:



An example of a taxonomy hierarchy

Note - 우리는 magnolis(목련)라 부르는 flower 클래스의 또다른 섭클래스를 만들지 않았다. 그것보다는 magnolia는 flower 클래스의 individual(instance) 이다. 왜 이런가? magnolia는 flower classification의 한 멤버이지만, 그것은 더 이상 flower subclassification이 아니다. 이것은 당연히 magnolia의 어의적 관점에서 보면 이것은 flower 클래스의 individuals(instances)이지 subclassification 즉, 다른 꽃이 아니라는 의미이다.

4.4 OWL Properties

OWL에서 Individuals은 *properties*와 관련 있다. OWL에는 두 종류의 property가 있다:

- **Object properties** (owl:ObjectProperty): 두 가지 OWL classes의 individuals (instances)와 관련이 있다.
- **Datatype properties** (owl:DatatypeProperty): OWL classes의 individuals (instances)의 literal values과 관련 있다.

An Example

먼저 a *data type* property (one which links an instance to a literal value)을 추가한 다음에, Magnolia가 속해 있는 *species family* 의 이름을 추가해 보자.

```
01. <rdf:RDF
02. xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03. xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
04. xmlns:owl="http://www.w3.org/2002/07/owl#"
05. xmlns:dc="http://purl.org/dc/elements/1.1/"
06. xmlns:plants="http://www.linkeddatatools.com/plants#">
07.
08. <!-- OWL Header Omitted For Brevity -->
09.
10. <!-- OWL Classes Omitted For Brevity -->
11.
12. <!-- Define the family property -->
13. <owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/plants#family"/>
14.
15. <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">
16.
17. <!-- Magnolia is a type (instance) of the flowers class -->
18. <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
19.
20. <!-- The magnolia is part of the 'Magnoliaceae' family -->
21. <plants:family>Magnoliaceae</plants:family>
22.
23. </rdf:Description>
24.
25. </rdf:RDF>
```

Important Point - 여기서, 만일 여러분이 object oriented programmer라면, 여러분은 마음으로부터 programmatic object classes and their associated properties를 생각해 낸 다음에, 그것들을 OWL classes에 대해 배운 것과도 비교하려 할 것이다. 그러지 마세요 - 정말 달라요. 위의 예제를 주목해 보자. 'family' property는 어떠한 class type과도 독립적이며, class flower (magnolia)의 instance에 할당된다. 똑같은 클래스의 또다른 instance 는 이

property를 갖지 않을 수 있다. OWL에서는 그러하다. instance를 가지고 있는 properties가 그것들의 class types이 아니라 그것들의 instance를 기술한다는 것을 주목하라. 이 경우에, 여러분은 완전히 서로 다른 클래스용으로 동일한 'family' property를 사용할 수 있다.

끝으로, an *object* property (one which links an instance to another instance)를 추가해 보자. 우리가 상점을 운영하고 있고 이 식물(Magnolia)을 상점주인과 마찬가지로 우리도 인기가 있는 다른 식물에 링크하길 원한다고 하자(Let's say we're running a shop, and we want to link this plant (Magnolia) to another plant which we know as the shop owner is equally as popular). "similarlyPopularTo"라는 property를 추가해 보자:

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:plants="http://www.linkeddatatools.com/plants#">

<!-- OWL Header Omitted For Brevity -->

<!-- OWL Classes Omitted For Brevity -->

<!-- Define the family property -->
<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/plants#family"/>

<!-- Define the similarlyPopularTo property -->
<owl:ObjectPropertyrdf:about="http://www.linkeddatatools.com/plants#similarlyPopularTo"/>

<!-- Define the Orchid class instance -->
<rdf:Description rdf:about="http://www.linkeddatatools.com/plants#orchid">

<!-- Orchid is an individual (instance) of the flowers class -->
<rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>

<!-- The orchid is part of the 'Orchidaceae' family -->
<plants:family>Orchidaceae</plants:family>

<!-- The orchid is similarly popular to the magnolia -->
<plants:similarlyPopularTordf:resource="http://www.linkeddatatools.com/plants#magnolia"/>

</rdf:Description>

<!-- Define the Magnolia class instance -->
<rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">

<!-- Magnolia is an individual (instance) of the flowers class -->
<rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>

<!-- The magnolia is part of the 'Magnoliaceae' family -->
<plants:family>Magnoliaceae</plants:family>

<!-- The magnolia is similarly popular to the orchid -->
<plants:similarlyPopularTordf:resource="http://www.linkeddatatools.com/plants#orchid"/>
```

```
42.
43. </rdf:Description>
44.
45. </rdf:RDF>
```

예를 들어, 우리는 URI http://www.linkeddatatools.com/plants#orchid와 더불어 Orchid를 표현하는 flowers 클래스의 새로운 individual(instance)을 정의하였다. 여러분이 우리가 동일한 클래스의 individual인 Mangnolia instance를 정의한 방법으로부터 이것을 이해할 수 있는지 확인해 봐라. 이제, 우리의 첫 번째 두 개의 tutorials에서처럼, 여러분이 위에서 정의한 RDF graph에 따라 Orchid 그리고 Magnolia class instances와 이것들의 predicates를 보여주는 그래프를 그릴 수 있는지 확인해 봐라.

Hint: 여러분은 양쪽 the *family* 그리고 *similarlyPopularTo* properties을 나타내는 화살표를 그려야 할 것이다. *family* property와 관련해서, 각각의 plant 용으로 서로 다른 family type literals을 포인트(point)해야 할 것이다. 그리고 *similarlyPopularTo* property와 관련해서는 the instances가 서로서로 포인트해야 할 것이다.

Important – Point Note
위의 예에서, 우리는 *similarlyPopularTo* object property를 통해 똑같은 OWL 클래스의 2 instances간을 링크하는 a two way를 갖는다(Orchid와 Magnolia 둘 다 똑같은 클래스의 individuals 이다). 그렇지만 주목할 것은 object properties는 two way가 아닐 수도 있다 – 그것들은 they may be one way일 수도 있다는 것이다. 그리고 중요한 것은 똑같은 OWL 클래스의 instances 간이 일어나지 않아야 한다(need not be between instances of the same OWL class). 그것들은 완전히 다른 OWL classes일 수 있다.

이제 이장을 마친다. 여러분은 다음과 같은 것을 이해하고 있어야 한다:

- That RDFS and OWL are W3C specifications with their own standard W3C namespaces.
- How OWL headers are constructed and some example uses.
- How to implement your own OWL classes, subclasses, individuals and properties.
- How to build your own basic ontology.

Tutorial 5: Querying Semantic Data

이제 여러분은 RDF에 시멘택스를 추가하는 방법, 그것을 출판하고 쿼리하는 방법을 알고 있

다. 관계형 데이터베이스의 테이블에서 SQL을 사용하여 쿼리하는 것처럼, RDF 데이터의 triples는 SPARQL을 사용하여 쿼리한다. 우리는 몇 가지 기본적인 쿼리를 조사하여 SPARQL과 SQL을 비교해 본다.

After this tutorial, you should be able to:

- Build your own basic SPARQL queries, step-by-step
- Understand that SPARQL is not just a query language, it is also a protocol
- Understand the XML format in which SPARQL protocol returns the results of queries
- Try executing some SPARQL queries yourself on some UK government RDF data
- Look ahead to SPARQL+ which has add, modify and delete capabilities

MS SQL or MySQL과 같은 관계형 데이터베이스로부터 데이터를 검색하는데 사용하는.
만일 여러분이 전통적인 IT에 대한 배경을 갖고 있다면, 여러분은 이미 MS SQL or MySQL과 같은 관계형 데이터베이스에서 데이터를 검색하는데 사용하는 SQL(Structured Query Language, pronounced "sequel")에 대하여 잘 알고 있을 것이다.

비슷하게, RDF data stores 역시 자신들의 쿼리 언어인 SPARQL (SPARQL Protocol and RDF Query Language, pronounced "sparkle")을 사용하여 쿼리한다. 그렇지만 SPARQL은 좀 더 고급스럽다.

SPARQL은 W3C standard 이며, 현재 버전은 1.1 이다.

5.1 A Starting Example

이제 예를 통해 SPARQL의 모습을 보여주기로 한다:

```
1. PREFIX sch-ont:   <http://education.data.gov.uk/def/school/>
2. SELECT ?name WHERE {
3. ?school a sch-ont:School.
4. ?school sch-ont:establishmentName ?name.
5. ?school sch-ont:districtAdministrative
          <http://statistics.data.gov.uk/id/local-authority-district/00AA>.
6. }
7. ORDER BY ?name
```

여러분은 이 모습에 익숙치않지만 걱정하지 마라.

이 쿼리가 만일 UK government's Open Semantic Database에서 수행된다면, 영국에서 행정구역 00AA에 있는 모든 학교의 이름을 얻게 될 것이다. 간단하게 쿼리를 살펴본 다음에, 이것을 정밀하게 조사하기 전에 여러분은 왜 이것에 대해 알아야 하는지를 생각해 보라.

Try It Yourself - Copy and paste the above query into the UK government's SPARQL query endpoint at http://education.data.gov.uk/sparql/education/query.html and the see the results. Such an URL is called a SPARQL endpoint in the semantic web world - and is the way in which the data on the semantic web is published to the outside world in a query enabled form.

SPARQL Is Similar To SQL

SQL처럼, SPARQL은 선택된 데이터의 어떠한 하위집합을 얻을 것인지를 결정하기 위하여 SELECT statement를 사용하여 query data set로부터 데이터를 선택한다. 또한 SPARQL 은 쿼리 데이터 세트에서 매취되는 것을 찾기 위한 그래프 패턴(graph pattern)을 정의하기 위하여 uses a WHERE clause을 사용한다.

SPARQL WHERE clause에서 graph pattern은 데이터에서 매치되는 것을 찾기 위하여, subject, predicate 그리고 object triple로 구성된다. 이제 위의 예를 좀 더 자세히 조사해 보자.

SELECT statement는 변수 ?name의 결과(returned)를 얻도록 한다.

Note - SPARQL에서, 변수 이름(variable names)은 question mark ("?") symbol을 접두사(prefix)로 갖는다. query graph pattern에서, 이것들은 어떤 node와 match 한다 - 그것이 resource 또는 literal이든 상관없다.

주목할 것은 이 변수 또한 WHERE clause search pattern에 사용될 수 있다 - second query search pattern의 object로. 그러나 또한 주목할 것은 ?school 변수이다. Because a 특정한 URI가 매치가 아니라 변수용이기 때문에, 어떠한 matching subject URI도 이 부분의 쿼리 패턴과 관련해서 리턴될 수 있으며 그 결과는 그 변수 이름과 mapped될 것이다.

그러므로, 위의 SPARQL query에서, ?name은 쿼리에서 지정한 3가지 탐색패턴과 매치되는 모든 학교이름을 리턴 시킨다. 우리가 원한다면, 우리는 추가적으로 매치 criteria를 추가함으로써 이 쿼리를 좀 더 전문화할 수 있다. 그렇지 않다면, 행정지역의 값 "00AA"과 매치되는 학교만을 요구하는 마지만 탐색 패턴을 제거함으로써, 우리는 위의 예에서 좀 더 포괄적인 결과를 얻을 수 있다.

마지막으로, 주목할 것은 ?school 변수는 모두 3가지인 탐색패턴과 관련해서, 어떤 subject가 탐색패턴과 매치되면 이 변수의 결과를 리턴시킬 것을 의미하고 있다. 그러나 이것이 이번 쿼리의 SELECT statement에 언급되지 않았으므로, ?school가 mapped되더라도, 그 결과 세트를 리턴하지는 않는다.

위에서 제시한 정부 데이터베이스에 대하여 쿼리해 보면 여러분 스스로 이것을 알게될 것이다.

5.2 SPARQL General Form

SPARQL queries 는 아래와 같은 일반적 행태를 가지고 있다. 이것은 하나의 쿼리는 여러 섹션으로 세분될 수도 있으며, 그 섹션을 정의하는 clause or keyword가 있다는 것을 보여주고 있다.

또는, 위에서 나타난 각 섹션의 예를 근거로, 우리는 다음과 같은 쿼리를 작성할 수 있다:

```
1. PREFIX plant:    <http://www.linkeddatatools.com/plants>
2. FROM <http://www.linkeddatatools.com/plantsdata/plants.rdf>
3. SELECT ?name WHERE {
4. ?planttype plant:planttype ?name.
5. }
6. ORDER BY ?name
```

한 단계 한 단계 SPARQL query를 형성하는 방법을 쉽게 이해해 보자.

5.3 Building A SPARQL Query

어떤 트리플 데이터와 관련해서 SPARQL 쿼리를 구축하는 방법을 배우기 위하여, 하나의 예제 데이터 세트로 시작해 보자: 우리가 이미 앞 강의에서 정의했던 plants & shrubs 용 온톨로지를 근거로 한다.

Example triple data containing a variety of shrubs and plants, and their family names

Subject	Predicate	Object
http://www.linkeddatatools.com/plants#magnolia	http://www.linkeddatatools.com/plants#family	Magnoliaceae
http://www.linkeddatatools.com/plants#african_lilly	http://www.linkeddatatools.com/plants#family	Liliaceae
http://www.linkeddatatools.com/plants#silvertop	http://www.linkeddatatools.com/plants#family	Aralianae
http://www.linkeddatatools.com/plants#velvetleaf	http://www.linkeddatatools.com/plants#family	Malvaceae
http://www.linkeddatatools.com/plants#manglietia	http://www.linkeddatatools.com/plants#family	Magnoliaceae

우리의 example data set에는 5가지의 plants & shrubs가 들어있다. subject는 그 식물을 표현하는 URI이다(가독성을 위하여 URI는 식물이나 관목의 일반 이름을 사용하였다). The predicate는 family name이다 - 이것은 우가 정의했던 온톨로지에서 가져왔다. 그런 다음에 우리는 literal objects를 사용한다 - 이것은 식물이나 관목의 과학적 과이름(family name)인 문자열로 구성되어 있다.

이 데이터와 관련해서 몇 가지 간단한 쿼리를 작성해서 그 결과를 살펴보자.

Select All Data

먼저, 위의 데이터로부터 모든 plant URIs (subjects) 그리고 plant family names (literal-type objects)을 선택할 쿼리를 작성해 보자.

```
1. PREFIX plants: <http://www.linkeddatatools.com/plants>
2. SELECT * WHERE
3. {
4. ?name plants:family ?family
5. }
```

SQL과 마찬가지로 SPARQL에서 wildcard '*'는 모든 매치된 데이터를 결과세트로 리턴시킬 것이다. 이것이 무엇을 의미하는가? 우리가 두 개의 변수 ?name 그리고 ?family를 언급했으므로, 그 쿼리는 우리의 결과 세트에 이렇게 선언된 두 가지 쿼리변수가 mapped 되는 subjects, predicates or objects에 따라, 그 결과를 리턴시킬 것이다.

이렇게 우리는 쿼리를 정의하였고, 이제 그것을 실행해 보자. SPARQL은 단지 query language만이 아니며, 그것은 또한 프로토콜이며 여러분의 소프트웨어가 읽을 수 있는 특별한 schema에 결과를 리턴 시킨다. 결과 세트 중에서 가장 널리 사용되는 형태들 중의 하나가 XML result format 이다. 아래는 우리가 되돌아갈 XML format result set 이다.

```
01. <?xml version="1.0" ?>
02. <sparql xmlns="http://www.w3.org/2005/sparql-results#"
03.  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
04. <head>
05. <variable name="name"/>
06. <variable name="family"/>
```

```
07. </head>
08. <results>
09. <result>
10. <binding name="name">
11. <uri>http://www.linkeddatatools.com/plants#magnolia</uri>
12. </binding>
13. <binding name="family">
14. <literal>Magnoliaceae</literal>
15. </binding>
16. </result>
17. <result>
18. <binding name="name">
19. <uri>http://www.linkeddatatools.com/plants#african_lilly</uri>
20. </binding>
21. <binding name="family">
22. <literal>Liliaceae</literal>
23. </binding>
24. </result>
25. <result>
26. <binding name="name">
27. <uri>http://www.linkeddatatools.com/plants#silvertop</uri>
28. </binding>
29. <binding name="family">
30. <literal>Aralianae</literal>
31. </binding>
32. </result>
33. <result>
34. <binding name="name">
35. <uri>http://www.linkeddatatools.com/plants#velvetleaf</uri>
36. </binding>
37. <binding name="family">
38. <literal>Malvaceae</literal>
39. </binding>
40. </result>
41. <result>
42. <binding name="name">
43. <uri>http://www.linkeddatatools.com/plants#manglietia</uri>
44. </binding>
45. <binding name="family">
46. <literal>Magnoliaceae</literal>
47. </binding>
48. </result>
49. </results>
50. </sparql>
```

이것은 매우 읽기 쉽고 양이 많다. 우리는 결과 세트가 사용하는 어떤 XML namespaces를 정의하는 사용할 수 있는 root <sparql> node를 갖고 있다. 그런 다음, 우리의 SELECT statement에서 wildcard (*)를 사용했기 때문에, WHERE clause 에 있는 쿼리 그래프 패턴에서 선언된 두 개의 변수인 ?name 과 ?family를 검색한다. 알다시피, 이러한 것들은 결과 세트의 <head> section에서 정의된다. 즉, 이것들을 쿼리에 의해 리턴되는 변수들이다.

이제 우리는 이들 쿼리 변수에 묶여진 결과를 얻는다. 각각의 패턴 매치는 그것 자체의

<result> node에 리턴된다. 주목할 것은 우리의 쿼리의 결과에서 어떠한 매치도 이루어지지 않았다면 우리는 우리의 XML에 리턴되는 어떠한 결과도 없다는 것이다.

여러분은 우리의 result XML이 데이터를 enclose하기 위하여 <uri> and <literal> tags를 사용함으로써 URI type values과 literal values을 구분하는 방법을 알 수 있다. 이것은 여러분이 필요로 할 때 이러한 구분법을 이용하여 이러한 데이터를 조사(parse)하도록 만들어진 소프트웨어에 의존하게 될 것이다. 그러나 그것들이 이러한 방법을 타이프한다는 사실은 리턴된 결과가 사용하는 시스템이 무엇이든지 간에 매우 유용하다.

Select Only "Magnoliaceae" Family

보다 정밀하게 우리의 데이터 세트로부터 Magnoliaceae family에 속하는 식물만을 리턴시켜 보자. 어떻게 할까?

```
1. PREFIX plants: <http://www.linkeddatatools.com/plants>
2. SELECT * WHERE
3. {
4. ?name plants:family "Magnoliaceae"
5. }
```

이것은 간단하게 이전의 쿼리에서 ?family 변수를 제거한 다음에, 그것 대신에 literal "Magnoliaceae"를 대치시키면 된다. 이제 우리의 쿼리는 어떤 object match보다 이 문자에 맞는 정확한 매치를 수행할 것이다.

쿼리 수행 결과는 다음과 같다:

```
01. <?xml version="1.0" ?>
02. <sparql xmlns="http://www.w3.org/2005/sparql-results#"
03. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
04. <head>
05. <variable name="name"/>
06. </head>
07. <results>
08. <result>
09. <binding name="name">
10. <uri>http://www.linkeddatatools.com/plants#magnolia</uri>
11. </binding>
12. </result>
13. <result>
14. <binding name="name">
15. <uri>http://www.linkeddatatools.com/plants#manglietia</uri>
16. </binding>
17. </result>
18. </results>
19. </sparql>
```

WHERE 조건절에서 ?family 변수를 제거했기 때문에, 이것은 결과에 더 이상 존재하지 않으며, 우리의 <head> section에는 단지 return variable인 ?name, 그리고 우리의 두 가지 matching triple patterns만이 포함 된다 - object value으로 문자값 "Magnoliaceae"를 가지고 있는 두 멤버 모두 다 Magnoliaceae family 이다.

5.4 Further Exploration

이번에는 여러분에게 무엇이 querying triple data와 같은지에 대하여 알아보자. Building SPARQL queries를 구축하는 것은 다소 실무적이지만, 알아야할 가치가 있다. 이제 여러분은 SPARQL query가 어떤 형태로 되어 있는지에 대해 잘 알았을 것이다. 추가로 더 많은 자료를 읽을므오써 여러분이 실습하도록 노력해라.

다행스럽게도, W3C에서 이 강의 내용을 넘어서서 이 주제에 대한 훌륭한 개론을 소개하고 있다. 여러분이 이것을 배우길 추천한다. 여러분은 W3C의 SPARQL Query Language For RDF 페이지에서 이것을 찾을 수 있다.

Note - You may like to try SPARQL queries of your own using our linked data tool suite RDF Studio. RDF Studio has full SPARQL help (including autocomplete of your queries) to help you write queries as you type.

5.5 What About CREATE, INSERT, UPDATE?

여러분이 triple data store로부터 데이터의 subsets을 얻기 위하여 SPARQL의 SELECT-WHERE 형태에 대한 기본들을 배우는 동안, 우리는 아직까지 CREATE, INSERT or UPDATE 방법들을 다루지 않았다. 이렇게 한 이유는 바로 그것들은 현재 실행되지 (implemented) 않기 때문이다.

만일 여러분이 그것에 대하여 생각한다면, 어느 정도 이해하여야 한다. 질의가능하고, 공식적인 SPARQL endpoint에서 데이터를 출판하는 단체들은 아마도 대부분의 경우에 자신들의 가치있는 데이터를 기술하거나 변화시키는 것을 원치 않을 것이다. 또한 어느 단계에서 그것은 missing requirement 같이 여겨지고 있다 - 특히 만일 여러분이 웹을 통해서 공식적이라기보다는 로컬리하게 triple data stores를 사용하고 있고 쿼리언어(SQL 데이터베이스에서처럼)를 사용하여 변경하고자 할 경우에. 어떻게 이것이 이뤄질 수 있을까?

현재 new specifications such as SPARUL (SPARQL/Update) and SPARQL+와 같은 새로운 스펙이 이런 문제를 해결하기 위하여 개발 중에 있지만, 기능적으로 이러한 문제를 해결할 수 있는 확실한 것(solid contender)는 아직 이 글을 쓰고 있는 기간에는 나타나지 않았다.

As RDF data가 그렇더라도 널리 채택되고 있더라도, Semantic Web frameworks에 의해 설

치되도록 등장할 a winning contender를 기대하자. 우리는 물론 관심을 기울일 것이다.

이 장을 마치면서, 여러분은 다음과 같은 것을 이해할 수 있어야 한다:

- That SPARQL can query RDF datasets, rather like SQL can query a relational database.
- That SPARQL endpoints are used to query and return data from the semantic web.
- How you can build and execute simple SPARQL queries yourself.
- Have a starting knowledge of the form in which SPARQL queries return results.

< 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