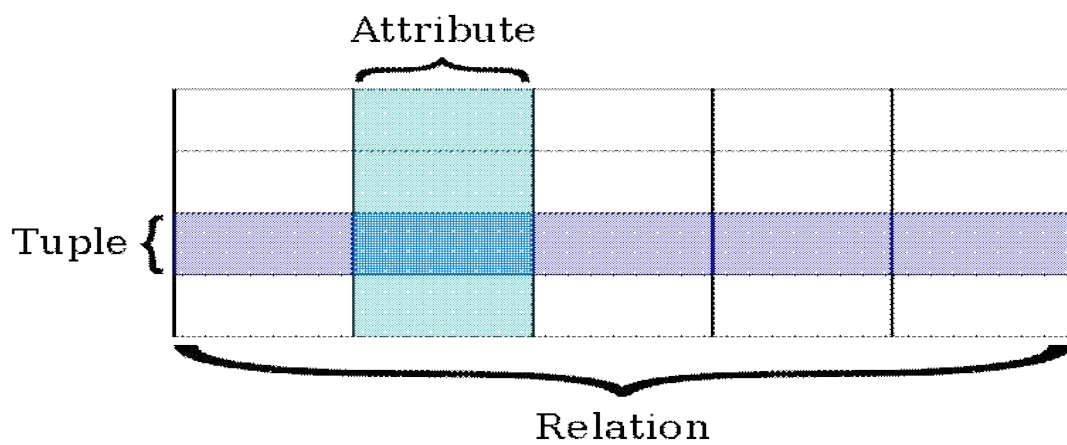


Database



2019

대구대 사회과학대학 문헌정보학과

CONTENTS

1. Terminology and overview
2. Applications and roles
3. History
4. Database type examples
5. Database design and modeling
6. Database languages
7. Database Normalization
8. Performance, security, and availability

<부록>

<Linked Data and The Semantic Web>

<List of Academic Databases and Search Engines>

1. Terminology and overview

데이터베이스란 조직화된 데이터의 집단이다. 데이터는 전형적으로 정보요구절차를 지원하기 위한 적절한 형태로 조직된다. 예를 들어, 도서관의 대출절차를 지원하기 위해 소장자료의 이용가능성을 모델링하는 것.

DBMS는 데이터의 수집 분석을 위해, 다양한 이용자, 어플뿐만 아니라 데이터베이스 그 자체와 상호작용이 가능한 어플들로 구성된다. 범용 DBMSs는 데이터베이스의 정의, 제작, 질문, 갱신, 그리고 운영이 가능하도록 디자인된 소프트웨어 시스템들이다. 잘 알려진 DBMS로는 MySQL, MariaDB, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, SAP, dBASE, FoxPro, IBM DB2, LibreOffice Base 그리고 FileMaker Pro 등이 있다. 데이터베이스가 다른 DBMS로 수출할 수 없는 것이 일반적이지만, 서로 다른 DBMS라 하더라도 복수의 데이터베이스와 작업이 가능한 SQL, ODBC, JDBC와 같은 표준 어플을 사용하면 이러한 문제를 해결할 수 있다.

1) SQL(Structured Query Language)

a relational database management system (RDBMS)에서 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 Java programming language용의 API 이다.

4) Application Programming Interface (API)

이것에서는 소프트웨어 구성요소들이 서로 어떻게 상호작용하여야 하는지를 밝히고 있다. 하드 디스크 드라이브나 비디오 카드 같은 데이터베이스나 컴퓨터 하드웨어의 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 대부분의 관심사를 가장 간단하고도 가장 기본적으로 모델화할 수 있기 때문이다. 또한 object-oriented 또는 semi-structured models, 최근의 graph data models, 그리고 XML과 같은 또 다른 모델의 결과들은 종종 관계형 모델의 결과에서 유래되기 때문이다.

공식적으로 “데이터베이스”란 데이터 그 자체와 구조를 지원하는 소프트웨어를 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. 따라서 “DBMS”란 이용자가 하나 이상의 데이터베이스와의 접촉할 수 있도록 하는 한 별(suite)로 구성된 컴퓨터 소프트웨어이다. 이 두 용어는 매우 밀접하게 서로 연관되어 있기 때문에 “데이터베이스”란 용어를 무심코 사용할 경우, 종종 DBMS와 그것이 관리하는 데이터 둘 다를 의미하기도 한다.

전문정보기술 분야를 벗어나면, 데이터베이스란 용어는 때때로 종종 어떤 데이터의 집합을 말하기도 한다(예: 스프레드시트, 또는 카드색인). 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 어플의 처리를 통한 정보의 획득

4) 관리

이용자의 등록과 감시, 데이터 보안 강화, 성능 감시, 데이터의 순수성 유지, 병행통제 처리, 시스템 문제시 데이터의 회복.

이외에도 DBMS는 저장된 데이터의 순수성과 안전성을 유지하고, 시스템에서 오류가 발생할 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.

데이터베이스와 DBMSs는 그것들이 지원하는 데이터베이스 모델(예, 관계형이나 XML이나), 기동 가능한 컴퓨터의 종류(server에서부터 휴대전화까지), 데이터베이스 접근용인 쿼리 언어(예, SQL or XQuery), 그리고 performance, scalability(확장성), resilience(복원), 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. Applications and roles

오늘날 선진국의 대부분 기관들은 자신들의 업무활동을 데이터베이스에 의존하고 있다. 점차적으로, 데이터베이스들은 그 기관의 내부업무를 지원하는데 사용될 뿐만 아니라, 고객과 업자 간의 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 트랜잭션 DBMSs는 많은 이용자가 동시에 그것에 질문하여 갱신될 때 고성능, 이용성, 보안성을 유지하기 위해 더욱 복잡해지고 있다. 그렇지만 개인용이면서 데스크 탑인 데이터베이스 시스템은 복잡성이 다소 떨어지는 경향이 있다. 예를 들어, 개인용 Microsoft Access와 FileMaker Pro는 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. 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 사용하여야 하는 시스템에 대한 많은 예가 존재한다. 한 가지 일반적인 예는 이메일시스템이다: 이메일시스템은 이메일 메시지를 최대로 잘 처리하도록 디자인되었으며, 범용 DBMS의 다양한 기능성을 크게 필요로 하진 않는다.

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

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

1) a wire protocol

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 layered 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, 저장 그리고 네트워크, 데이터베이스의 크기, 용량, 성능과 그것들을 다루는 각각의 DBMSs가 규모면에서 크게 성장하고 있다. 데이터베이스 기술의 발전은 데이터 모델이나 구조에 따라 3 세대로 구분할 수 있다: navigational, SQL/relational, post-relational. 두 가지 주요한 초기 네비게이션 데이터 모델은 계층 모델(hierarchical model)들이며, Codasyl model (Network model)은 이 구조의 전형이다.

1) navigational database

다른 사물로부터 나온 레퍼런스를 우선적으로 추적하여 레코드와 사물을 찾을 수 있는 데이터베이스의 한 종류이다. 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(불러오다) 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.

Codd에 의해 1970년에 처음으로 제안된 관계형(relational) 모델은 어플이 연결된 링크보다는 콘텐츠에 의해 데이터를 탐색해야 한다는 모델이므로, 기존의 전통적 모델과는 차이가 났다. 관계형 모델은 회계장부형태의 테이블(ledger-style tables)로 구성되어 있으며, 각각의 테이블은 서로 다른 entity용으로 사용되었다. 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 뿐

만 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

이 데이터베이스는 관계형 데이터베이스에서 사용된 tabular 관계보다는 다른 수단으로 모델화된 데이터의 저장 및 검색 메커니즘을 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.

4. Database type examples

데이터베이스를 분류하는 첫 번째 방법은 그것들의 콘텐츠의 종류, 예를 들어 서지, 문서-텍스트, 통계, 또는 멀티미디어 사물과 관련짓는 것이다. 두 번째 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.

1) in-memory 데이터베이스

기본적으로 주 메모리에 상주하고 있는 데이터베이스이지만, 전형적으로 비휘발성 컴퓨터 데이터 저장매체에 의해 백업된다. 주 메모리 데이터베이스는 디스크 데이터베이스보다 훨씬 빠르며, 그래서 통신 네트워크 장비에서 응답시간이 중요할 때 종종 사용된다. SAP HANA 플랫폼은 in-memory 데이터베이스용으로 매우 뜨거운 주제이다.

(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 architecture를 갖는다. 잠재적 용도 들로는 보안 감시, 경고, 통계, 수집과 인증 등이 있다. 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 있는 “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.

(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.

4) 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 are more expressive than relational databases but less expressive than logic programming systems.

(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.

(3) Prolog is a general purpose logic programming language associated with artificial intelligence and computational linguistics. Prolog was one of the first logic programming languages, and remains the most popular among such languages today.

5) distributed 데이터베이스

데이터와 그것의 DBMS를 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.

6) document-oriented 데이터베이스

도큐먼트 위주 또는 반-정형화된 데이터의 정보를 저장, 검색, 관리하도록 디자인되었다. 도큐먼트-중심 데이터베이스는 NoSQL 데이터베이스의 주요 유형 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 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.

7) graph 데이터베이스

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

(1) A **triplestore** is a purpose-built database for the storage and retrieval of triples, 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.

8) 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) **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.

9) 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.

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 유닛이 자신만의 주 메모리를 갖고 있지만, 모든 유닛들이 다른 저장공간을 공유한다.
- c) Shared nothing architecture: 아무것도 공유하지 않는 구조: 각각의 프로세싱 유닛이 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.

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, 이 정보를 수집할 수 있어야 할 것이다.

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.

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.

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.

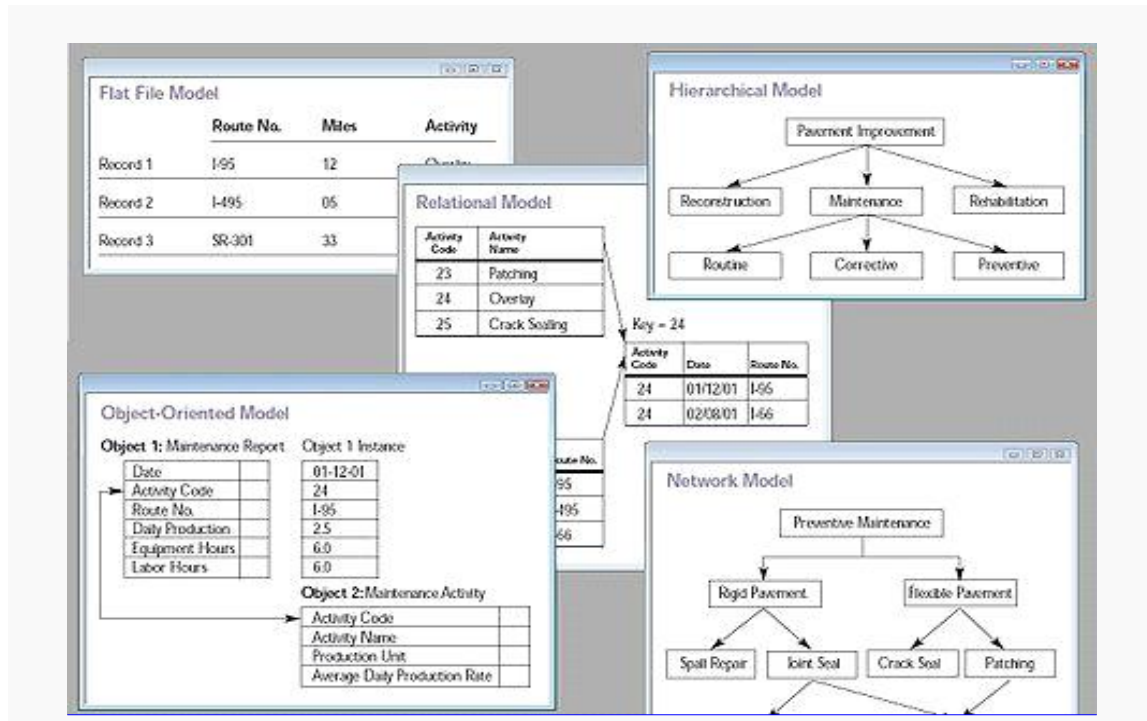
1) **Database normalization**

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.

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.

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.

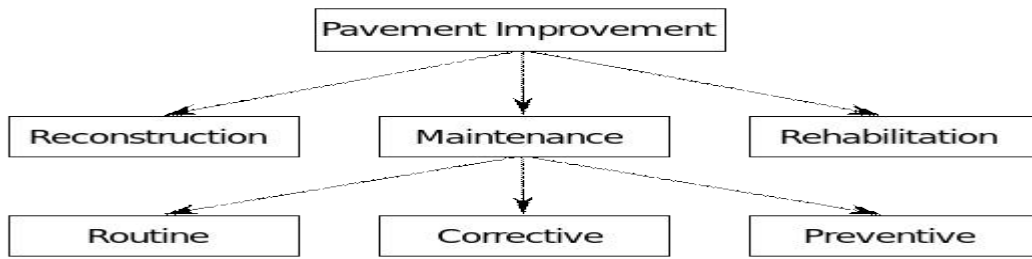
데이터베이스 모델이란 기본적으로 어떠한 방식으로 데이터를 저장, 조직, 취급할 것인지를 다루는 데이터베이스의 논리적 구조를 결정하며 데이터 모델의 한 유형이다. 가장 인기 있는 데이터베이스 모델은 테이블을 근거로 하는 형식의 관계형 모델(또는 관계형과 유사한 SQL 모델)이다.

데이터베이스용의 일반적인 논리적 데이터 모델에는 다음과 같다:

1) Hierarchical database model

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

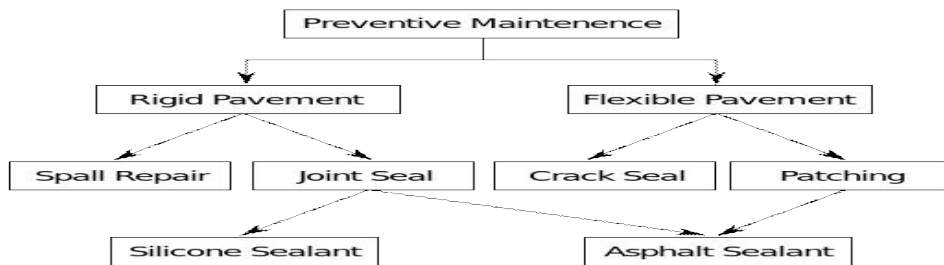
Hierarchical Model



2) Network model

네트워크 모델은 사물과 그것들의 관계를 표현하는데 있어서 유연한 방법으로 인식되고 있는 데이터베이스 모델이다. 이것의 뛰어난 특징은 그 스키마(사물 유형은 nodes로, 관계 유형은 아크로 표현된 그래프와 같은)가 계층구조든 격자(lattice)구조든 상관하지 않는다는 것이다.

Network Model



3) Relational model

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

Relational Model

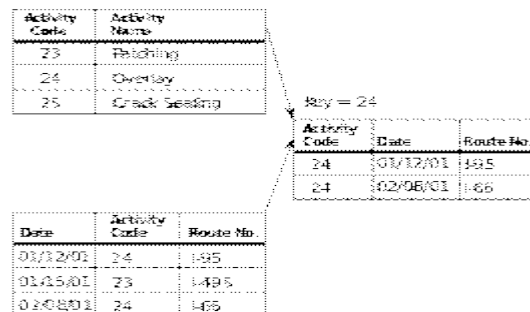
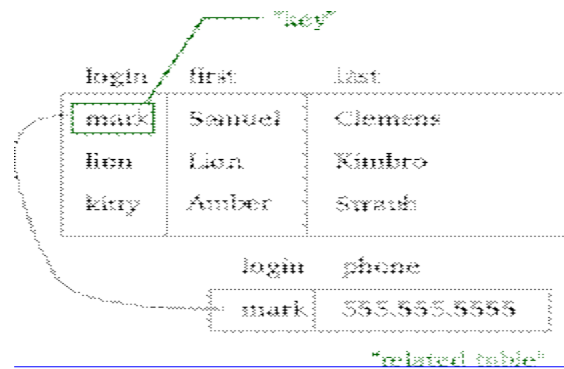


Diagram of an example database according to the Relational model.



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

대부분의 관계형 데이터베이스는 SQL 데이터 정의와 쿼리 언어를 사용한다.

4) Entity-relationship model

ER 모델은 데이터베이스를 설명하는 하나의 추상적 방법이다. 테이블에 데이터가 저장되는 관계형 데이터베이스의 경우에, 테이블에 있는 데이터는 다른 테이블에 있는 데이터와 관계를 갖는다. - 예를 들어, 데이터베이스에 있는 여러분의 인적 항목(entry)은 여러분 자신의 전화번호와 연결되어 표현할 수 있다. ER 모델에서 여러분은 하나의 객체이고, 각 전화번호도 하나의 객체이며, 여러분과 전화번호와의 관계는 'has a phone number' 이다. 이 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 구조적 내용 (granular detail)을 포함한다. 개념적 ER 모델에서는 보통 대상기관에서 공통으로 사용하는 master reference data를 객체로 정의하고 있다.

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 모델에서 master data entities에 대한 구조적 공통성(commonality)을 확립하는 것이다.

(4.2) Logical data model

논리적 ER 모델은 개념적 ER 모델보다 더 많은 details를 포함한다. master data entities에 따라서, 다양한 객체가 정의된 다음, 각 객체의 details를 개발한다. 그런 다음에 이 객체간의 관계를 결정한다. 논리적 ER 모델은 그것을 실행시킬 수 있는 테크놀로지와는 독립적으로 개발되어야 한다.

(4.3) Physical data model

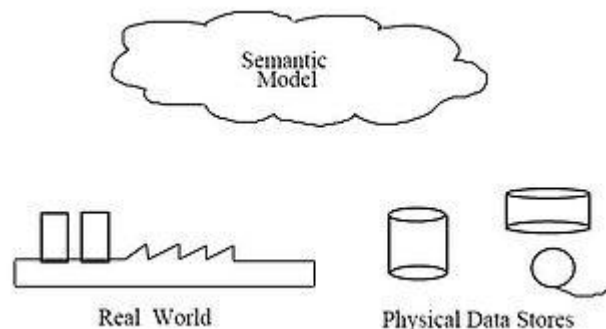
한 개 이상의 물리적 ER 모델이 각각의 논리적 ER 모델로부터 개발될 수 있다. 물리적 ER 모델은 보통 하나의 데이터베이스처럼 개발되고 있다. 그러므로 각각의 물리적 ER 모델은 데이터베이스를 생산할 수 있는 충분한 details를 포함하고 있어야 하며, 각각의 물리적 ER 모델은 각각의 데이터베이스 관리 시스템과 다소 차이가 있더라도 기술적으로 독립성을 가져야 한다.

5) Semantic model

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

썬텐틱 모델은 fact-oriented 이다(객체 지향적의 반대). 사실은 데이터 요소간의 이진관계에 의해 전형적으로 표현된다. 반면에 고차원의 order 관계는 이진 관계의 집단으로 표현된다. 전형적으로 이진관계는 triples의 형태인 Object-Relation Type-Object로 이루어진다: (예) the Eiffel Tower <is located in> Paris.

Klas와 Schrefl(1996)에 따라, “썬텐틱 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

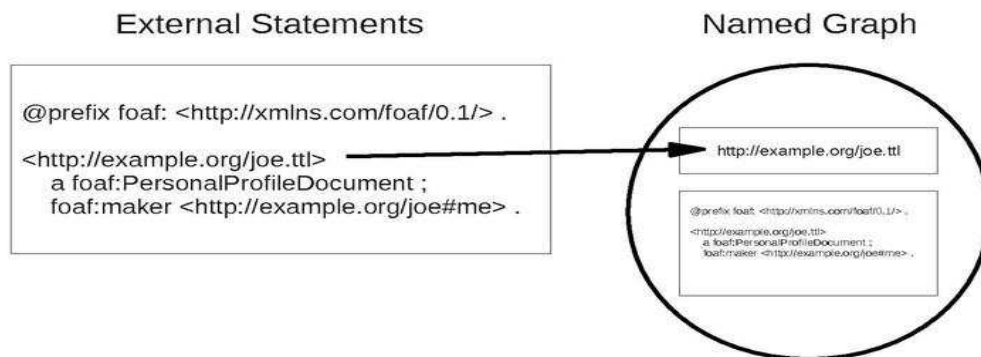
XML 데이터베이스는 데이터를 XML 포맷에 저장할 수 있는 data persistence software system 이다. 이렇게 저장된 데이터는 원하는 포맷으로 queried, exported, 그리고 serialized될 수 있다. XML 데이터베이스는 대체로 다큐먼트 지향적 데이터베이스와 결합한다.

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.")

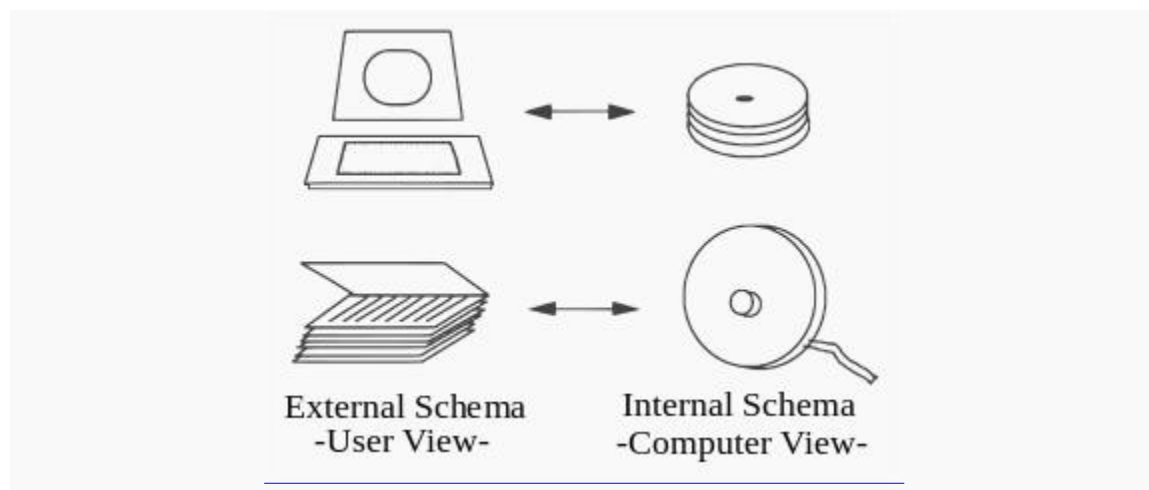
7) Named graph

Named graphs는 한 세트의 Resource Description Framework statements(a graph)를 URI를 사용하여 식별하는 Semantic Web 구조의 중요한 개념이다. 이 URI에는 context, provenance (기원, 출처) 정보, 또는 기타 메타데이터와 같은 statements에 대한 address description을 포함한다. Named graphs는 graph를 제작할 수 있는 RDF 데이터 모델을 간단하게 확장시킨 것이지만, 이 모델은 일반적으로 일단 웹으로 출판되면 그것들을 구별하는 효과적인 수단으로는 부족하다.



Describing a named graph

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 구조 모습을 다룬다.

이러한 3 차원의 데이터베이스 구조는 관계형 모델의 중요한 기본 능력의 하나인 데이터 독립성과 관련이 있다. 이 아이디어는 어느 수준에서 일어난 변화는 그 보다 고차원에 있는 뷰에는 영향을 끼치지 않는다는 것이다.

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 1986년에 ANSI의 표준이 되었으며, 1987년에 ISO의 표준이 되었다.

2) 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.

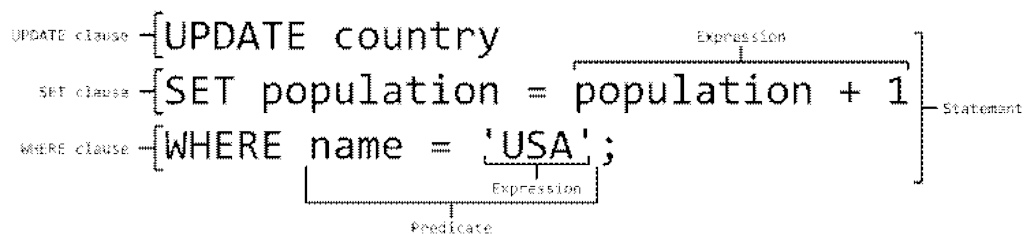
3) SQL/XML

SQL/XML은 XQuery와 SQL이 결합된 것이다.

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.

6.2 Syntax



6.3 Language elements

SQL 언어는 다음과 같이 여러 가지의 언어요소들로 세분된다:

1) Clauses

statements 와 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 statements와 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 플랫폼에서 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.

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 return해야 한다는 것을 특정화하기 위하여 사용될 수 있다. SELECT는 SQL에서 그것에 포함되는 선택적 키워드와 절(clauses)과 더불어 가장 복잡한 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 사용되

어야 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 사용되느냐를 밝힌다. ORDER BY 절이 없다면, SQL에서 얻어진 로우들의 순서는 정의할 수 없다(undefined).

다음은 비싼 책들의 리스트를 보여주는 SELECT 쿼리의 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 붙인 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 동지화(nested)될 수 있다. 동지화된 쿼리를 subquery라 부른다. joins와 다른 테이블 연산자들이 컴퓨터에서 많은 경우에 superior(다시 말해서, faster) 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;
```

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 포함(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

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

```
TRUNCATE TABLE example;
```

?
?
?

4) DROP

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

```
DROP TABLE example;
```

?
?
?

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 디지털의 숫자를 결정하는 실제적 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) 디지털들의 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.

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;

8. Database normalization

데이터베이스 정규화(Database normalization)는 데이터 중복(redundancy)을 최소화하기 위하여 관계형 데이터베이스의 컬럼과 테이블을 조정(분할)하는 과정이다. 정규화에서는 대체로 한 개의 커다란 테이블을 보다 작은 복수의 테이블로 쪼갠 다음에, 그것들 간의 관련성을 설정한다. 정규화의 목적은 어떤 컬럼의 추가, 삭제, 변경이 단지 한 개의 테이블에서만 이루어지고 있다면, 이미 정의된 규칙에 따라 나머지 다른 테이블에까지 그 영향력이 유전된다는 것이다.

관계형 모델의 개발자인 Edgar F. Codd가 정규화의 개념을 소개하였으며, 1970년에 First Normal Form(1NF)을, 1971년에 Second Normal Form (2NF)과 Third Normal Form (3NF)을, 그리고 Codd와 Raymond F. Boyce가 1974년에 Boyce-Codd Normal Form (BCNF)을 정의하였다. 비공식적으로, 관계형 데이터베이스 테이블은 그것이 제 3 정규형에 있으면, “정규화”되었다고 말하기도 한다. 대부분의 제 3 정규형의 테이블은 삽입(insertion), 갱

신(update), 그리고 삭제(deletion) 이상(anomalies)에서 자유롭다.

7.1 Purpose

1970년에 Edgar Frank "Ted" Codd가 정의한 the first normal form의 기본적인 목적은 first-order logic을 바탕으로 하는 “a universal data sublanguage”를 사용하여, 데이터를 질의하고 조작하기 위한 것이다.

1) First-order logic uses quantified variables over non-logical objects. It allows the use of sentences that contain variables, so that rather than propositions such as Socrates is a man one can have expressions in the form X is a man where X is a variable

2) data sublanguage

A language or part of a language concerned only with database query and update and/or database definition.

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

1. 원하지 않는 삽입, 갱신, 삭제의 의존성으로부터 관계의 집단(the collection of relations)을 자유롭게 하는 것;
2. 새로운 유형의 데이터를 소개함으로써 관계의 집단의 재구성에 대한 필요성을 인식하여, 응용 프로그램의 life span을 늘리는 것.
3. 관계형 모델을 이용자에게 보다 정보적으로(informative) 만드는 것;
4. 관계의 집단을 query statistics - 이 통계들은 시간이 지남에 따라 변화에 취약하다 - 에서 중립적으로 만드는 것(균형을 이루도록 하는 것이다)..

7.1.1 Free the database of modification anomalies

Employees' Skills

Employee ID	Employee Address	Skill
425	87 Spymore Grove	Typing
425	67 Spymore Grove	Shorthand
519	134 Chatham Street	Public Speaking
519	135 Market Avenue	Cooking

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 부작용(side-effects)이 발생한다. 모든 테이블이 이러한 부작용으로 어려움을 겪는 것은 아니다; 부작용은 충분하게 정규화되지 않은 테이블에서만 발생한다. 잘못 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 Employee ID, Employee Address, Skill 속성을 갖고 있을 수 있다; 그러므로 특정한 종업원의 주소의 변경은 잠재적으로 복수의 레코드에 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 문제를 발생시킨다. 이러한 현상을 **update anomaly**라 한다.

2) 어떤 사실이 결코 기록될 수 없는 환경이 존재한다.

예를 들어, "Faculty and Their Courses" 테이블의 레코드는 Faculty ID, Faculty Name, Faculty Hire Date, Course Code의 속성들을 갖고 있다. 그러므로 우리는 적어도 한 과목을 가르치는 교수에 대한 내역을 기록할 수 있으나, 아직 과목을 배정받지 못한 신입교수에 대한 내역을 기록할 수 없다. 왜냐하면, Course Code 속성이 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.

위의 예인 "Faculty and Their Courses" 테이블은 이러한 이상(anomaly)을 겪고 있다. 만일 교수에게 임시로 어떤 과목의 배정을 중지한다면, 우리는 효과적으로 그 교수의 교과목만을 삭제하여야 하지만, 이 테이블의 경우에 교수 레코드의 과거 값들이 모두 삭제된다. 이러한 현상을 **deletion anomaly**라 한다.

7.1.2 Minimize redesign when extending the database structure

새로운 유형의 데이터를 받아들이기 위하여 정규화된 데이터베이스 구조를 확대하고자 할 때, 기존에 존재했던 데이터베이스 구조는 거의 또는 전혀 변하지 않아야 한다. 결과적으로, 그 데이터베이스와 상호작용하는 어플들에게는 영향을 최소화 하여야 한다.

7.1.3 Make the data model more informative to users

정규화 테이블은 실세계의 개념과의 상호연관성을 반영하여야 한다(mirror).

7.1.4 Avoid bias towards any particular pattern of querying

정규화 테이블은 일반용의 쿼리에 적합하여야 한다. 즉, 내역을 예상할 수 없는 미래의 쿼리를 다루는 테이블에서도 모든 쿼리가 사용가능해야 한다는 것이다. 그렇지만 비정규화 테이블은 어떤 종류의 쿼리에만 적합(lend)하다.

7.2 Example

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

Customer	Transactions		
Jones	Tr. ID	Date	Amount
	12890	14-Oct-2003	-87
	12904	15-Oct-2003	-50
Wilkinson	Tr. ID	Date	Amount
	12898	14-Oct-2003	-21
Stevens	Tr. ID	Date	Amount
	12907	15-Oct-2003	-18
	14920	20-Nov-2003	-70
	15003	27-Nov-2003	-60

반복적인 거래 (a *repeating group* of transactions)가 각각의 고객에게서 나타나고 있다. 그러므로 고객의 거래에 관한 쿼리는 크게 다음과 같은 2 단계로 평가 된다:

- 1) 거래 집단을 고객 별로 해체한다(unpacking);
- 2) 1 단계의 결과를 근거로 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.

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 반영될 수 있다; 반면에 이전의 상황에서는 어떤 값들은 특별하게 처리해야만 하는 하위 단계의 구조에 포함되어 있었다. 따라서 정규화 디자인은 범용적 쿼리를 처리할 수 있지만, 비정규화(unnormalized) 디자인은 그렇게 하지 못한다.

7.3 Background to normalization: definitions

7.3.1 Functional dependency(함수적 의존성)

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

7.3.2 Full functional dependency(완전한 함수적 의존성)

만일 어떤 속성이 1) 함수적으로 X에 의존하고 있으나 2) X의 어떤 하위 집합에 함수적으로 올바르게 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, $X \rightarrow Z$ 는 단지 $X \rightarrow Y$ 와 $Y \rightarrow Z$ 에 의해서만 이루어지는 의존성을 말한다.

7.3.4 Trivial(하찮은, 사소한, 일상적) functional dependency

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

7.3.5 Multivalued dependency(복수값 의존성)

복수값 의존성이란 테이블의 어떤 로우가 어떤 다른 로우들의 존재를 암시하는 제한적 요소 (constraint)를 말한다.

7.3.6 Superkey(슈퍼키)

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

7.3.7 Candidate key(후보키)

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

예를 들어, 컬럼 <Name>, <Age>, <SSN> and <Phone Extension>의 로우로 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).

7.3.8 Non-prime attribute(비- 으뜸 속성)

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

7.3.9 Prime attribute(으뜸 속성)

으뜸 속성은 역으로 어떤 후보키에서만 발생하는 속성이다.

7.3.10 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.

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 같이 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 전화번호-유형의 도메인으로 정의되어 있다고 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 로우에 복수의 값을 허용함으로써 이것은 1NF를 위반하고 있다. 전형적인 관계형 데이터베이스 관리 시스템에서는 한 테이블 안에 위와 같이 복수의 값을 포함하는 필드들을 허용하지 않는다.

7.4.1.2 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 Telephone Number

<u>Customer ID</u>	<u>First Name</u>	<u>Surname</u>
123	Robert	Ingram
456	Jane	Wright
789	Maria	Fernandez

<u>Customer ID</u>	<u>Telephone Number</u>
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 레코드를 생산한다. key인 고객 ID를 사용하면, 일-대-다의 관계가 두 테이블 간에 존재한다. “부모” 테이블인 고객 이름에 있는 로우는 “자녀” 테이블인 고객 전화번호에 있는 많은 전화번호 로우를 가질 수 있다. 그러나 각각의 전화번호는 단지 하나의 고객에게만 속한다. 따라서 주목해야 하는 것은 이 디자인을 통해 2 그리고 3 NF의 필수조건을 충족시킬 수 있다는 것이다.

7.4.1.3 1NF tables as representations of relations: 관계의 표현으로서 1NF 테이블

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].

7.4.2 Second normal form

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

요약하면, 어떤 테이블이 1NF에 있고 그것의 모든 비-오피스 속성이 후보키에 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 한번 이상(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(나무깎기)**” 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		Employees' Skills	
<u>Employee</u>	Current Work Location	<u>Employee</u>	<u>Skill</u>
Brown	73 Industrial Way	Brown	Light Cleaning
Harrison	73 Industrial Way	Brown	Typing
Jones	114 Main Street	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

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

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

이것의 근본적인 문제는 Winner Date of Birth 속성이 의존하고 있는 전이적 의존성 (transitive dependency) 때문이다. Winner Date of Birth는 실제로 Winner에 의존하며, 그 다음에 키 Tournament / 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 dependencies를 갖지 않아야 한다.

다음의 테이블에는 복수의 후보키가 나타나 있다:

Electric Toothbrush Models

<u>Manufacturer</u>	<u>Model</u>	<u>Model Full Name</u>	<u>Manufacturer Country</u>
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 하위집합에 적합하게 의존하고 있다: Manufacturer. 이 디자인을 2NF로 변경하기 위해서는 다음과 같은 2 개의 테이블이 필요하다:

Electric Toothbrush Manufacturers

<u>Manufacturer</u>	<u>M a n u f a c t u r e r</u> <u>Country</u>
Forte	Italy
Dent-o-Fresh	USA
Kobayashi	Japan
Hoch	Germany

Electric Toothbrush Models

<u>Manufacturer</u>	<u>Model</u>	<u>Model Full Name</u>
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 (1) 엔티티가 2NF에 있고, (2) 테이블의 모든 속성은 단지 으뜸키에만 확실하게 의존한다는 참조무결성(Referential integrity)의 법칙을 지키도록 하여, 데이터의 중복성을 줄이도록 정규화하는 3번째 디자인 과정이다.

7.4.3.1 Definition of third normal form

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 $X \rightarrow Y$ 그리고 $Y \rightarrow Z$ (그렇지만, $Y \rightarrow X$ 인 경우가 아니다)이지만, 간접적으로 $X \rightarrow Z$ (X가 Z를 결정한다)인 함수적 의존성을 말한다.

7.4.3.2 "Nothing but the key"

3NF의 조건을 충족시키지 못하는 2NF의 예는 다음과 같다:

Tournament Winners

<u>Tournament</u>	<u>Year</u>	<u>Winner</u>	<u>Winner Birth</u>	<u>Date of</u>
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 알려야하기 때문에, 복합키 {Tournament, Year}는 어떤 로우를 유일하게 식별하기 위하여 필요한 최소한의 속성 집합이다. 즉, {Tournament, Year}는 이 테이블의 후보키이다.

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

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

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

이제 3NF에 있는 두 개 모두의 테이블에서 갱신이상(更新異常)이 일어날 수 없다.

7.4.4 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.

7.4.5 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 유

형의 이상(anomaly) 발생에 초점을 맞추고 있다.

만일 관계형 스키마가 BCNF에 있다면, 비록 현재 다른 유형의 과잉이 존재하더라도 함수적 의존성에 의존하고 있는 모든 과잉은 제거된다. 그리고 단지 그것의 의존성 $X \rightarrow Y$ 의 모든 것과 관련해서 적어도 다음과 같은 조건들 중에서 하나를 유지하고 있다면, 그 관계형 스키마 R은 BCNF에 있다고 말한다:

- 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 테이블은 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(회원)
1	11:00	12:00	SAVER(회원)
1	14:00	15:30	STANDARD(비회원)
2	10:00	11:30	PREMIUM-B(비회원)
2	11:30	13:30	PREMIUM-B(비회원)
2	15:00	16:30	PREMIUM-A(회원)

- 테이블의 각 row는 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:

- $S_1 = \{\text{Court, Start Time}\}$
- $S_2 = \{\text{Court, End Time}\}$
- $S_3 = \{\text{Rate Type, Start Time}\}$
- $S_4 = \{\text{Rate Type, End Time}\}$
- $S_5 = \{\text{Court, Start Time, End Time}\}$
- $S_6 = \{\text{Rate Type, Start Time, End Time}\}$
- $S_7 = \{\text{Court, Rate Type, Start Time}\}$
- $S_8 = \{\text{Court, Rate Type, End Time}\}$

- $S_T = \{\text{Court, Rate Type, Start Time, End Time}\}$, 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 서로 다른 두 개의 예약이 동시에 시작되거나 동시에 끝난다는 것을 주목하여야 한다. 이것이 왜 $\{\text{Start Time}\}$ 과 $\{\text{End Time}\}$ 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 $S_1 \subset S_5$ 이므로 S_5 는 후보키가 될 수 없기 때문이다. 2NF에서 후보키에 대한 비-오픈 속성의 부분적 함수적 의존성을 금지하고 있다는 것과 3NF에서 후보키에 대한 비-오픈 속성의 전이적 함수적 의존성을 금지하고 있다는 것을 기억하라.

Today's Court Bookings 테이블에서 어떠한 비-오픈 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, $\text{Rate Type} \rightarrow \text{Court}$ 의존성 때문이다. $\text{Rate Type} \rightarrow \text{Court}$ 의존성은 Rate Type가 단지 단일 Court에만 적용되어야 할 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 $\{\text{Rate Type}\}$ 과 $\{\text{Court, Member Flag}\}$ 이다: Today's Bookings 테이블의 후보키들은 $\{\text{Rate Type, Start Time}\}$ 그리고 $\{\text{Rate Type, End Time}\}$ 이다. 이들 둘 다 BCNF에 있다. $\{\text{Rate Type}\}$ 이 Rate Types 테이블의 키일 때, 두 개의 서로 다른 코트와 연결된 Rate Type을 갖는 것은 불가능하다. 그러므로 Rate Types 테이블에 있는 키로서 $\{\text{Rate Type}\}$ 을 사용함으로써, 최초의 테이블에 영향을 끼치는 이상(anomaly)을 제거할 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 행정가에게 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.

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.

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(downside)은 최초로 갱신된 데이터베이스 데이터와 그것들을 동기화시키고자 갱신할 때 발생하는 추가비용(overhead)과 메모리의 여분에 대한 경비(cost)이다.

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

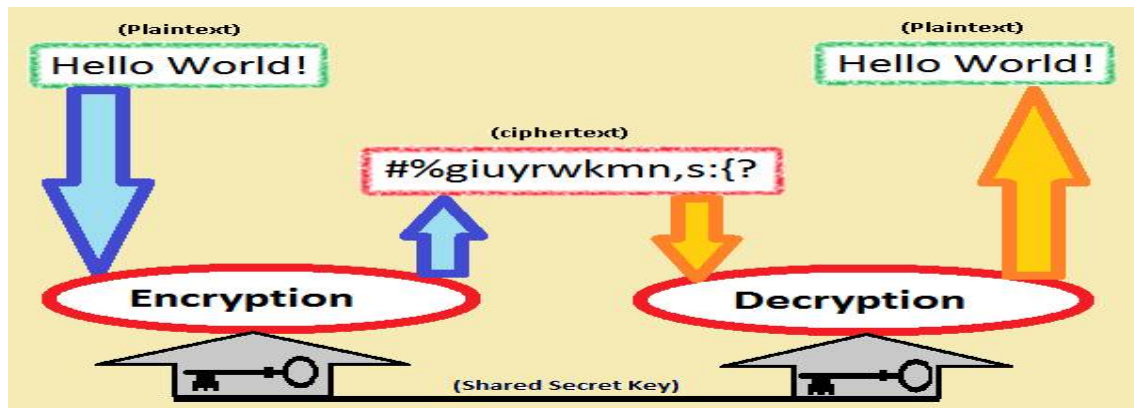
데이터베이스 보안이란 데이터베이스의 콘텐츠, 소유자, 이용자를 보호하는 모든 요소를 다루는 것이다. 이것의 범위는 고의적이고 불법적인 데이터베이스 사용으로부터의 보호에서부터 불법 객체(예, 사람이나 컴퓨터 프로그램)에 의한 비고의적인 접근까지이다.

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 급여 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.

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

2) **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.

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.

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 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 사용가능해야 한다. 따라서 데이터베이스의 개념적 그리고 외적 구조 수준은 그러한 변형 안에서도 유지되어야 한다.

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 의해 필요한 이용자 인터페이스를 제공하고 있다.

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 인터페이스

의 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.

8.6 Backup and restore

때때로 여러 가지 이유로 데이터를 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.

<부록>

< Linked Data And The Semantic Web >

<<http://www.linkdatatools.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 다큐먼트에 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. 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 다큐먼트. 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 다큐먼트 which the user knows about and is trying to get to. Rather, the user is trying to locate a number of 다큐먼트s 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 다큐먼트들이며, 서로 하이퍼링크로 연결되어 있다. 사람이나 기계 모두 이 같은 다큐먼트를 읽을 수 있다. 그렇지만, 사람은 전형적으로 페이지에서 키워드를 찾을 수 있지만, 기계가 이 다큐먼트에 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 데

이터베이스의 웹 사이트가 만들려 한다면, 그것의 HTML 다큐먼트들은 해당 데이터로만 만들어져야 한다.

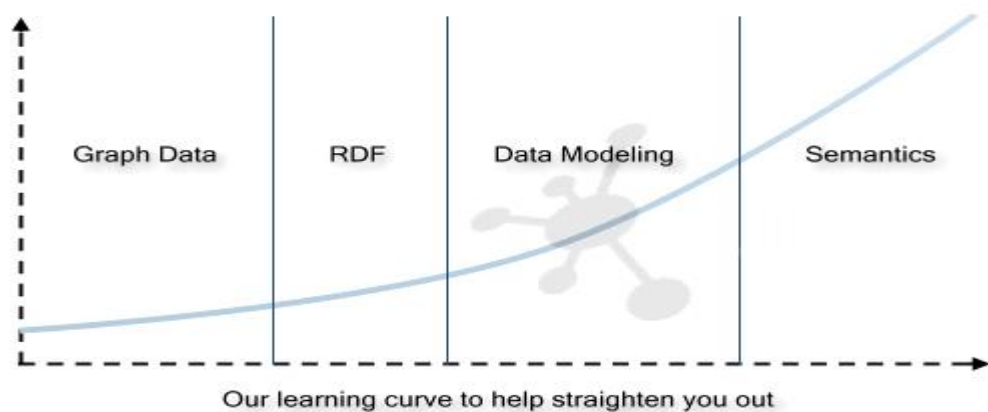
따라서 SW는 이 같은 문제를 해결하기 위하여, 다음과 같은 여러 가지 방법으로 현재의 인터넷 모습을 변화시키고 있다:

- 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)

실제로, 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, SW와 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?

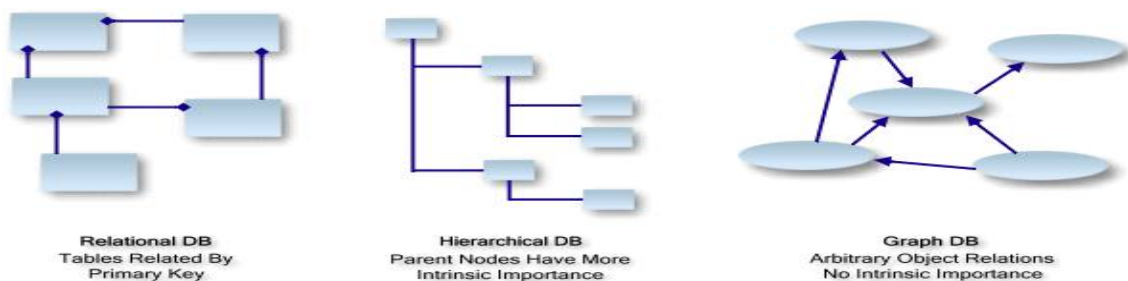
- To state the precise meaning of (a word or sense of a word, for example).
- To describe the nature or basic qualities of; explain:

비록 이 데이터베이스에 대해 여러분이 생소하더라도, 이것이 바로 전 세계적으로 SW를 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 다큐먼트를 보자. 전형적으로 XML 다큐먼트에는 한 개의 parent node와 함께 여러 개의 정보 노드를 사용하고 있다. 따라서 이러한 다큐먼트의 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.

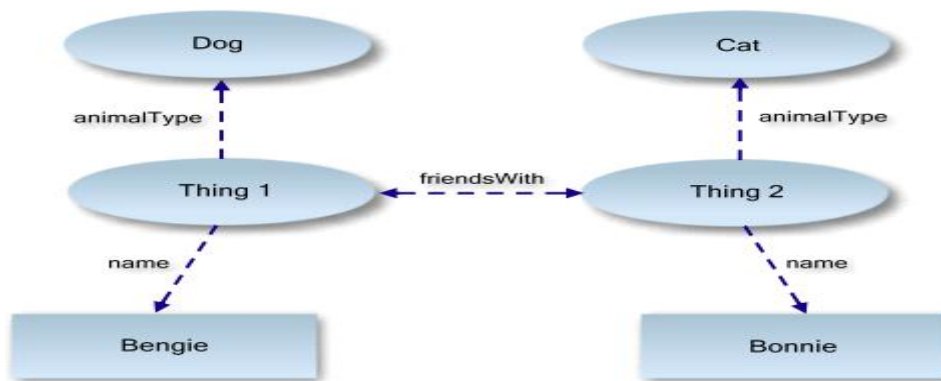
An Example Of A Data Graph

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 디자인 된 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 아이টে를 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

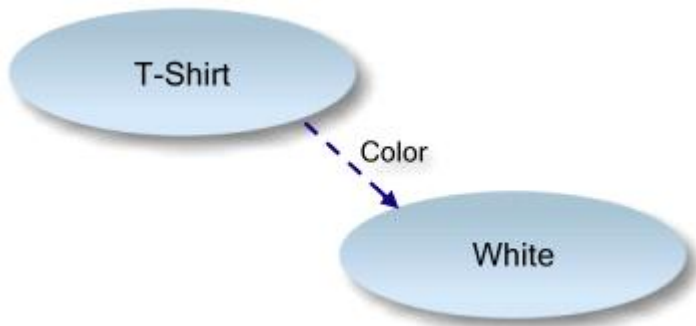
```
01. <?xml version="1.0" encoding="UTF-8"?>
02.
03. <rdf:RDF
04.     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
05.     xmlns:dc="http://purl.org/dc/elements/1.1/"
06.     xmlns:region="http://www.country-regions.fake/">
07.
08.     <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Oxford">
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 콘텐츠에서 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 용어인데, 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 다큐먼트의 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) 다큐먼트가 RDF 다큐먼트라는 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 System은 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 위키피디아의 이름이다.

위의 다큐먼트에서 namespace를 표현하고 있는 RDF node가 바로 이 RDF 다큐먼트의 roots node이다.

>Add A Statement

RDF 다큐먼트에는 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 대해서는 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:resource`는 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 본문으로 돌아가서, `<feature:color>`의 속성은 3번째 줄에 있는 `xmlns:feature`에 이미 포함되어 있다는 것도 이해하여야 한다. 다시 말해서, `xmlns:feature`의 이름리스트에는 이미 `size`와 `color`라는 속성이름이 포함되어 있다는 것이다.

!!! RDF 다큐먼트에 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#white`인 지시문을 참조하고 있는 객체와 더불어 `feature:color`이라는 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, `literal`과 `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 보라:

- Subject of the statement?

- Predicates of the statement(including whether they are resources or literals)?
- Objects referenced by the **resource** predicates?

이제 RDF 문서들에 대해 이해했고, 그것이 데이터 그래프와 어떤 관계가 있는지를 알았다면, 여러분은 RDF graph data로 시멘틱스를 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-shirt`처럼 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 문서 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.

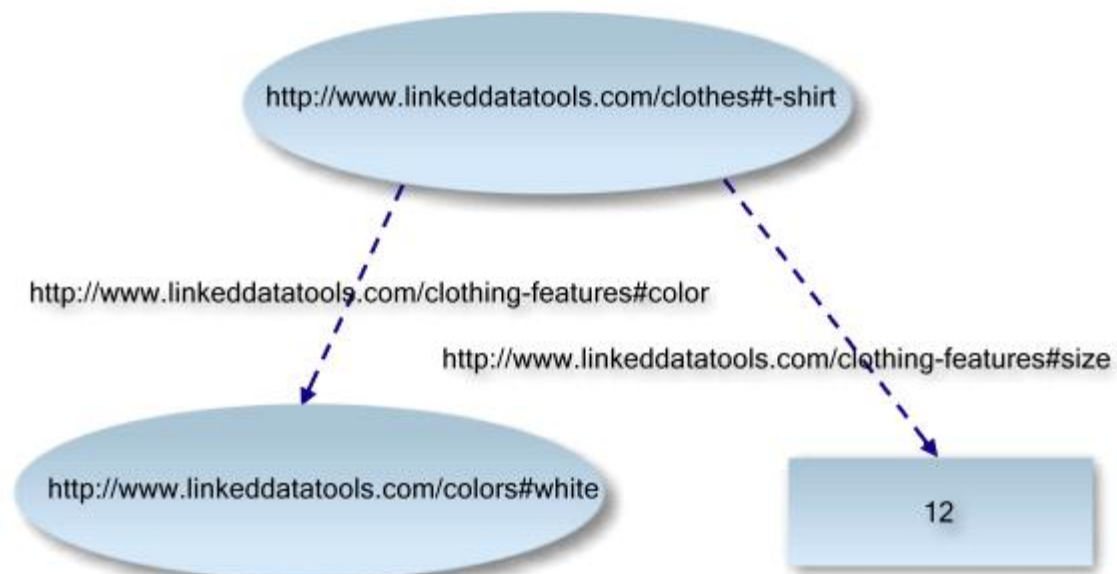
이 namespace의 목적은 단지 동일한 이름의 태그로 인하여 발생하는 이름간의 충돌을 피하기

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#size>로 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 다큐먼트s in RDF/XML
- Understand how and why RDF uses URIs to identify subjects, predicates and objects
- How to relate an RDF 다큐먼트 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	Semantics (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
Hierarchical	XML	Tag/Attribute Values	XSD/DTD	e.g. Unique Attribute Key Value	XPath	N/A
Graph	RDF/XML, Turtle	RDF	RDFS/OWL	URI	SPARQL	Yes, using RDFS and OWI

*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) 다큐먼트.

*DTD?

A 다큐먼트 type definition (DTD) is a set of markup declarations that define a 다큐먼트 type for an

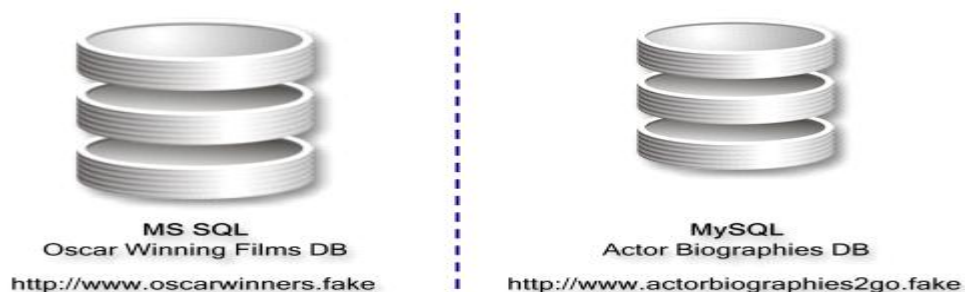
SGML-family markup language (SGML, XML, HTML). A 다큐먼트 Type Definition (DTD) defines the legal building blocks of an XML 다큐먼트. It defines the 다큐먼트 structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML 다큐먼트, or as an external reference.

*XPath?

XPath, the XML Path Language, is a query language for selecting nodes from an XML 다큐먼트. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML 다큐먼트. 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 가치를 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, 시멘틱스를 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.fake>에 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 콘텐츠에

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 콘텐츠에는 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.fake>의 이용자는 출연배우의 이름을 클릭하면, 그들에 대해 더 많은 정보를 찾을 수 있다는 이점이 있다. - 이 정보는 <http://www.actorbiographies2go.fake>의 MySQL database에 저장되어 있다.

- 똑같이, <http://www.actorbiographies2go.fake>의 이용자가 출연배우들의 영화이름을 클릭하여 더 많은 정보를 얻을 수 있다. 이것은 <http://www.oscarwinners.fake>에 있는 MS SQL database에 저장되어 있다.

- 두 사이트간의 어떠한 데이터 공유도 자신들의 데이터베이스에 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 콘텐츠를 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 이루어진 않는다.

2) Sharing With The Semantic Web Model

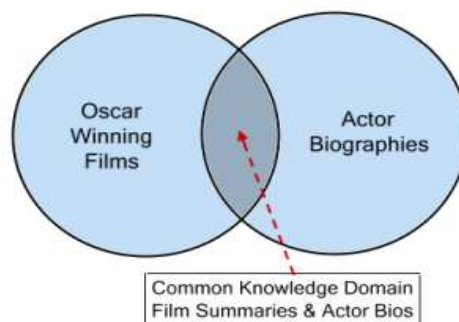
시멘틱 모델링에서, 먼저 다음과 같은 중요한 용어에 대해 이해하여야 한다:

- 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, 시멘틱 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, 촬영 장소와 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 데이터 베이스에서 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 신택스인 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 문서의 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 문서이다.

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 다큐먼트 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 보라. 이것은 Dublin Core Metadata Initiative라는 namespace를 참조하며, machine readers에게 **dc:title** and **dc:description** 와 같은 엘리먼트들은 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 클래스를 슈퍼클래스로 가지고 있는 **planttype** class. 따라서 **planttype** class는 모든 plant types의 최상위 클래스이다.

```
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 Class Definition - Plant Type -->
11. <owl:Class rdf:about="http://www.linkeddatatools.com/plants#planttype">
12.
13.   <rdfs:label>The plant type</rdfs:label>
14.   <rdfs:comment>The class of all plant types.</rdfs:comment>
15.
16. </owl:Class>
17.
18. <!-- OWL Subclass Definition - Flower -->
19. <owl:Class rdf:about="http://www.linkeddatatools.com/plants#flowers">
20.
21.   <!-- Flowers is a subclassification of planttype -->
22.   <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#planttype"/>
23.
24.   <rdfs:label>Flowering plants</rdfs:label>
25.   <rdfs:comment>Flowering plants, also known as angiosperms.</rdfs:comment>
26.
27. </owl:Class>
28.
29. <!-- OWL Subclass Definition - Shrub -->
30. <owl:Class rdf:about="http://www.linkeddatatools.com/plants#shrubs">
31.
32.   <!-- Shrubs is a subclassification of planttype -->
33.   <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#planttype"/>
34.
35.   <rdfs:label>Shrubbery</rdfs:label>
36.   <rdfs:comment>Shrubs, a type of plant which branches from the base.</rdfs:comment>
```

```

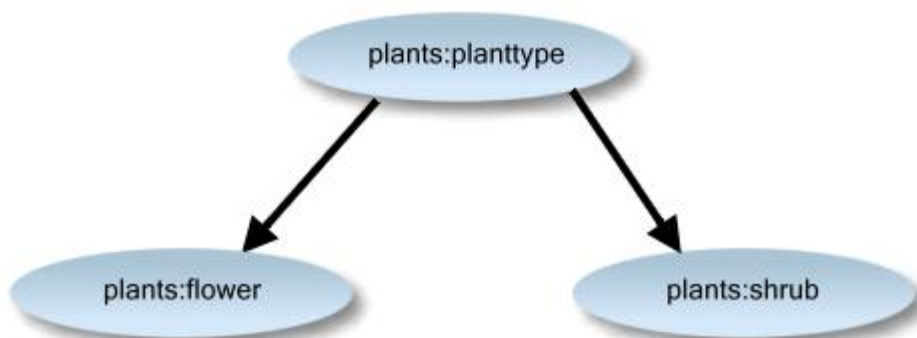
37.
38. </owl:Class>
39.
40. <!-- Individual (Instance) Example RDF Statement -->
41. <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">
42.
43. <!-- Magnolia is a type (instance) of the flowers classification -->
44. <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>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

```

Point Of Interest(RDF 다큐먼트 의 타당성 조사) - You can investigate this RDF 다큐먼트 further by passing it through W3C's [RDF validator](#), which automatically tabulates the 다큐먼트'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 섹클래스를 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:

```
01. <rdf:RDF
02. xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03. xmlns:owl="http://www.w3.org/2002/07/owl#"
04. xmlns:dc="http://purl.org/dc/elements/1.1/"
05. xmlns:plants="http://www.linkeddatatools.com/plants#">
06.
07. <!-- OWL Header Omitted For Brevity -->
08.
09. <!-- OWL Classes Omitted For Brevity -->
10.
11. <!-- Define the family property -->
12. <owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/plants#family"/>
13.
14. <!-- Define the similarlyPopularTo property -->
15. <owl:ObjectPropertyrdf:about="http://www.linkeddatatools.com/plants#similarlyPopularTo"/>
16.
17. <!-- Define the Orchid class instance -->
18. <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#orchid">
19.
20. <!-- Orchid is an individual (instance) of the flowers class -->
21. <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
22.
23. <!-- The orchid is part of the 'Orchidaceae' family -->
24. <plants:family>Orchidaceae</plants:family>
25.
26. <!-- The orchid is similarly popular to the magnolia -->
27. <plants:similarlyPopularTordf:resource="http://www.linkeddatatools.com/plants#magnolia"/>
28.
29. </rdf:Description>
30.
31. <!-- Define the Magnolia class instance -->
32. <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">
33.
34. <!-- Magnolia is an individual (instance) of the flowers class -->
35. <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
36.
37. <!-- The magnolia is part of the 'Magnoliaceae' family -->
38. <plants:family>Magnoliaceae</plants:family>
39.
40. <!-- The magnolia is similarly popular to the orchid -->
41. <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#orchid>와 더불어 Orchid를 표현하는 flowers 클래스의 새로운 individual(instance)을 정의하였다. 여러분이 우리가 동일한 클래스의 individual인 Mangnolia instance를 정의한 방법으로부터 이것을 이해할 수 있는지 확인해 보라. 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 보라.

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 간에 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 시멘텍스를 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 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 SPARQL은 쿼리 데이터 세트에서 매치되는 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"와 매치되는 학교만을 요구하는 마지막 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.

PREFIX (Namespace Prefixes)

e.g. PREFIX plant: <http://www.linkeddatatools.com/plants>

SELECT (Result Set)

e.g. SELECT ?name

FROM (Data Set)

e.g. FROM <http://www.linkeddatatools.com/plantsdata/plants.rdf>

WHERE (Query Triple Pattern)

e.g. WHERE { ?planttype plant:planttype ?name }

ORDER BY, DISTINCT etc (Modifiers)

e.g. ORDER BY ?name

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 타입한다는 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 포함된다 - 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, 알아야 할 가치가 있다. 이제 여러분은 SPARQL query가 어떤 형태로 되어 있는지에 대해 잘 알았을 것이다. 추가로 더 많은 자료를 읽을수록 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 공식적이라기 보다는 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 database 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 Geostrophysical 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.

*** OAlster**

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, dissertations, 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.

*** SSRN: 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