목차

저작권 고지

문서 최종 갱신일

이 문서는 2014년 01월 07일에 최종 갱신되었습니다. 최신 문서는 http://databaser.net 에 있습니다.

피드백

문서에 대한 의견이나 오류는 admin@databaser.net 로 메일을 주시거나 http://databaser.net 에 글 남겨주시면 됩니다.

1장. 이론들

콜브(kolb)에 따르면 학습에는 [구체적 경험] -> [성찰(반성)적 관찰] -> [추상적 개념화] -> [능동적 실험]과 같은 사이클을 가진다고 한다. 이 싸이클은 경험에 대한 이론적 근거를 찾고 이해하여 형성된 개념이 현실에도 부합하는지 테스트 해보는 IT의 학습법인 '백문이불여일타'와 유사함을 알 수 있다.

우리는 학습할 때에 예제 코드를 직접 타이핑(구체적 경험)한다. 그리고는 코드를 분석(성찰적 관찰)하고, 그 코드에 담긴 의미가 무엇인지 확인(추상적 개념화)한다. 그리고는 예제 코드를 이렇게도 바꿔보고 저렇게도 바꿔(능동적 실험)본다.

추상적인 개념화가 없는 실무 경험은 단순한 현상 관찰로 끝나게 된다. 어떤 근거나 원리 또는 이론을 바탕으로 하지 않고 '내가 예전에 해봤는데...'를 주된 논리로 삼은 사람들을 보았을 것이다. 이 사람들은 과거의 방법으로 일을 하다가 과거와는 다른 현상(결과)을 관찰하게 되고는 혼란을 겪게 된다. 그리고는 이렇게 바꿔보고 현상을 관찰하고, 저렇게 바꿔보고 현상을 관찰하는 시행착오법으로 일을 진행한다. 하지만, 일이 잘 풀리지 않아 시행착오법이 계속되면서 나의 행위에 대해 의심을 하게 되고, 왜 그렇게 되는지 알 수 없으므로 할 수 없이 현상만을 쫓게 된다. 그래서 이런 사람들은 실패를 자주하거나 야근이 잦다. 물론 태도의 문제도 있다. 왜 그렇게 될 수 밖에 없었는지에 대한 탐구를 하지 않으면 현상에 대한 관찰만이 있을 뿐이다. 우연치 않게 한 번에 성공하거나 또는 많은 시행착오 끝에 성공을 했어도 콜브의 이론대로 추상적 개념화와 능동적 실행이 되지 않는다면 그저 현상 관찰로 끝나게 된다. 이런 경험이 더하면 더해질수록 점점 선무당이 되어가고 주위의 여러 사람들을 괴롭힌다.

나의 행위에 대한 이론적인 근거가 무엇인지 아는 것은 매우 중요하다. 이론적인 근거가 없는 경험에 의한 방법이 지금의 현재 상황에서 최적의 솔루션이 될 수 없는 이유는 단순하다. 과거의 상황과 현재의 상황이 다르기 때문이다. 역사적으로 증명된 이론을 근거로 하는 행위는 정당성을 인정받을 수 있으며, 상황이 변해도 상황에 맞추어 응용하여 행동을 할 수 있게 끔 해준다. 이론은 상황에 맞춰진 것이 아닌 여러 상황이 일반화된 것이기 때문이다. 또한 '왜 이 이 일을 해야 하는가?' 에 대한 질문의 답을 알고 있으므로 강력한 동기부여가 된다. 이는 곧 생산성의 큰 차이를 가져오게 된다.

흔히들 이론과 실무의 갭(gap)에 대해서 이야기하고는 한다. 특히나 이론의 실무 적용 문제는 어느 분야나 논란이 되는 부분이다. 혹자는 이론과 실무는 다르다고 말한다. 우리는 역사의 중간에 살고 있기 때문에 이미 잘 정리된 이론들과 이론이 정립되지 않은 실무적인 것들에 끼어서 바둥거리는 상황이다. 현재(2013년 9월) 시점에서 볼 때 잘 정리된 이론들은 자료구조, 알고리즘, 데이터베이스론 등과 같은 것들이고, 빅데이터와 같은 것들이 이론이 정립되지 않은 것들에 속한다. 이런 상황은 언제까지나 계속될 것인데, 선구자가 되지 못하면 변화에 재빨리 적응할 수 밖에 없는 것이 IT생태계의 상황이다.

요즘 데이터베이스 분야의 여러 아티클들에서 데이터 품질(data quality)이나 마스터 데이터 관리

(master data management), 데이터 거버넌스(data governance)와 같은 용어를 자주 접해 보았을 것이다. 이러한 용어가 등장하게 된 배경은 현실과 컴퓨터 세계의 갭에 의한 비효율이 발생하기 때문이다. 1장에서는 이러한 현실과의 갭이 왜 생기는지 알아보고, 어떻게 하면 현실과 컴퓨터 세계와의 갭을 줄일 수 있지는 알아볼 것이다. 또한 데이터베이스 구축에 필요한 이론들을 살펴보고 어떻게 실무에 적용해야 하는지 간단하게 살펴볼 것이다. 자세한 실무 응용은 2, 3장에서 살펴볼 것이다.

1.1 전산화

78954 * 6987 / 564. 이 사칙 연산은 초등학교만 졸업을 해도 답을 구할 수 있다. 암산하거나 손으로 풀 수 있다. 물론 정확성을 요구하는 계산이라면 검산도 해야 할 것이다. 이를 반복적으로 수행하는 것은 엄청난 비효율이다. 그래서 이러한 비효율을 없애고자 하는 생각에 기인하여 전자계산기가 탄생을 하게 되었다. 전자계산기를 이용할 경우 78954 * 6987 / 564를 정확하게 입력했는지 눈으로만 확인한다면 정확하고 신속하게 계산 결과를 알 수 있다. 이렇게 절약된 시간은 좀 더 생산적인 일에 투자할 수 있다.

전산화란, 현실 세계의 일부를 컴퓨터 세계로 옮기는 작업이다. 이 전산화의 정의는 데이터베이스 관점에서 다음의 2가지 의미가 내포되어 있다.

1. 작업의 범위와 명세
2. 데이터 품질(정확성)

전산화의 정의에서 굳이 '일부'라는 단어를 쓴 이유는 현실 세계의 모든 것을 컴퓨터 세계로 옮길 수는 없기 때문이다. 물론 현실 세계의 모든 것을 컴퓨터 세계로 옮길 필요는 없다. 필요 이상으로 컴퓨터 세계에 현실 세계를 반영시키면 비효율이 발생한다. 그렇기 때문에 요구사항을 조사하고 분석하는 것이다. 요구사항은 현실 세계를 얼마만큼 컴퓨터 세계로 옮기는지에 대한 명세다. 물론 말이나 문서로 생각을 전달받고, 요구사항 분석가는 자신이 가진 지식과 경험을 토대로 이해할 것이다. 여기서 갭이 발생한다. 커뮤니케이션 채널이 많아지면 많아질수록 더욱 더 많이 갭이 발생한다. 인력이 많이 투입하여도 좋은 결과를 얻지 못하는 것은 이런 이유다. 또한 요구사항이 무엇인지 모르고 일단 봐야 알겠다는 사용자도 많다. 아래의 그림은 이러한 갭을 설명해준다.

<그림 1.1.1>

대규모 프로젝트에서는 처음부터 요구사항을 100% 도출한다는 것은 신의 영역이라 불릴 만큼 어렵다. 하지만 상황에 맞게 여러 개발방법론과 개발 프로세스를 적용하면 비교적 초기 단계에서 사용자의 기대와 현실과의 갭을 많이 좁힐 수는 있다. 프로젝트는 시간이 지날수록 이러한 갭이 좁혀지는데, 프로젝트 막바지까지 원하는 수준까지 갭을 좁히지 못한다면 프로젝트는 실패로 끝나게 된다.

요구사항 조사/분석 단계에서 갭을 최대한 줄이게 된다면 프로젝트의 성공에 더욱 가까워지지만, 시간이 지날수록 갭을 줄이는 것은 어렵게 된다. 구현의 막바지에서는 기술적인 한계나 잘못된 설계 등의 이유로 갭을 줄이는 것은 더욱 어려워지므로 반드시 구현 전까지 최대한 갭을 줄여야 한다. 물론 프로젝트의 성격에 따라 적정 수준의 사용자의 기대치 미치도록 순환 형식으로 개발을 진행하는 방법도 있는데, 주로 정보계 프로젝트에서 많이 진행하는 방식이다.

현실 세계의 일부를 컴퓨터 세계로 옮기는 작업의 결과 즉, 전산화된 결과물은 현실과의 일치성이 결여되어서는 안 된다. '데이터 품질'의 저자 잭 올슨은 데이터 품질 요소에서 무엇보다도 '정확성'이 우선되어야 한다고 했다. 정확성이란, 현실 세계의 것이 컴퓨터 세계에 얼만큼 유사하게 반영되었는지에 대한 척도다. 예를 들어, 필자 이재학은 현실에서 남자다. 그러나 전산화된 결과에 이재학이 여자로 표현되어 있다면 정확성이 떨어지는 것이다.

사람은 언제나 실수를 한다. 전산화도 사람이 하는 일인지라 실수를 한다. 정확성이 떨어지는 데이터는 1백만 건 중에 1백 건이 있을 수 있고, 1천 건이 있을 수 있다. 여러 조직 계층의 사람들에게 정확성을 어느 정도 수준까지 허용할 것인지 물어보면 조직마다 심지어 같은 조직 안에서조차 많이 다르게 답변하는 경우가 많다. 독자들도 각자 기준이 다를 것이다. 그래도 이 정도면 대략 쓸 만 하다 싶은 허용 기준이 있을 것인데, 의사결정자들이 모여 합의한 그 기준이 '품질 수준'이다.

'6시그마'라는 용어를 들어보았을 것이다. 경영 기법 또는 경영 철학으로 소개되기도 하고, 품질 개선/유지 방법으로 소개되기도 한다. 여기서 '시그마'는 표준편차인데, 6시그마는 [평균±6표준편

차]를 말한다. 즉, 목표수준이 1백 만 건 중에 3.4건 정도만의 결함을 허용하는 수준(품질 수준)이다. 마찬가지로 정성적이든 정량적이든 데이터도 품질 수준이 있다. 사용자들이 큰 불만 없이 사용한다면 적당한 수준인 것이다. 하지만 보고서 간의 데이터가 불일치 된다거나, 서버 간의 시간 차이 계산상의 차이 등과 같은 문제로 인해 사용자가 불만을 갖게 되어 데이터가 사용되지 않는다면 수준 미달인 것이다. 물론 사용자가 요구하는 데이터가 없는 것도 불만에 포함된다.

실제 요구사항과 구현된(전산화된) 결과의 차이도 있지만, 모르거나 어렵거나 또는 귀찮아서 현실을 제대로 반영하지 않는 것도 큰 문제다. 자, 이제 여러분이 작업해 놓은 결과물을 보자. 현실을 컴퓨터 세계로 잘 옮겨 놓았는가? 잘한 것인지 못한 것인지도 모르겠는가? 모르겠다면 이 책을 잘 선택한 것이다. 이 책은 전산화를 잘하는 그 중에서도 데이터베이스를 잘 구축하는 이론과 방법을 설명하는 책이기 때문이다. 이제 전산화하는 방법들에 대해서 차근차근 살펴보도록 하자.

1.2 전산화 방법(데이터베이스 관점)

'전산화'라는 작업을 하려면 일단 현실 세계에 대한 이해가 선행되어야 한다. 인간들은 세상을 앎(이해)을 위해 개념을 사용한다. 개념은 추상화된 관념을 말하는데, 우리는 개념을 통해서 세상을 이해한다. 추상화는 대상들에서 공통이 되는 부분(속성 or 특성)을 추출하여 단순하게 만드는 것을 말한다. 이는 우리가 지식을 터득하는 방법과 같다. 세세한 부분까지 사진처럼 기억하지는 않다. 자, 독자들에게 묻겠다. 다음의 그림이 무엇들을 나타내가?

<그림 1.2.1>

그렇다. 우리는 위의 그림들이 모두 '시계'라는 것을 알고 있다. 위의 그림이 시계를 나타내는 그림이란 것을 모르는 사람은 없을 것이다. 하지만, '갓난 아기들은 모르지 않겠느냐' 라고 반문하는 사람들도 있을 것이다. 맞다. 갓난 아기들은 위 그림들이 무엇인지 모른다. 왜냐하면 시계에 대한 개념이 없기 때문이다.

데이터베이스가 필요한 전산화 과정 중이라면 데이터 모델링을 실시할 것이다. 독자들은 다음의 3가지 데이터 모델에 대해서도 들어보았을 것이다.

1. 개념적 데이터 모델
2. 논리적 데이터 모델
3. 물리적 데이터 모델

위 3가지 모델은 1 -> 2 -> 3의 순서로 만들어진다. 개념적 데이터 모델이 가장 먼저 만들어지는 이유는 업무에 대한 이해를 필요로 하기 때문이다. 앞서 언급한 대로 이해(앎)를 위해서는 '개념'

을 필요로 한다. 그러므로 개념적 데이터 모델은 업무에 대한 이해의 결과물 이라고 할 수 있겠다. 업무를 이해 한다는 것은 데이터 모델링의 대상이 되는 부분과 시스템 구축 범위의 인식을 할 수 있다는 의미도 된다.

개념적 데이터 모델을 만드는 데도 이론이 필요한데, 필자의 견해로는 형이상학(형체(形體)를 초월(超越)한 영역(領域)에 관(關)한 과학(科學)이라는 뜻으로, 철학(哲學)을 일컫는 말[1])의 실재론이 그나마 가장 가깝다고 보여지지만, 데이터베이스 이론 전체적으로는 유명론도 실재론도 모두 사용된 것으로 보여진다.

논리적 데이터 모델은 개념적 데이터 모델을 토대로 어떤 논리적인 구조로 변환시킨 데이터 모델이다. 여러 가지 논리적인 구조[2]가 있지만, 현재는 대부분 '관계형(relational) 데이터 모델'만을 사용한다.

관계형 데이터 모델은 또한 RDBMS(Relational DataBase Management System)에서 물리적으로 구현될 것이므로 논리적 데이터 모델에서 물리적 데이터 모델로 변환하는 것은 쉽다. 물리적 데이터 모델에서는 디스크의 배치, 인덱스 등의 물리적인 객체에 대한 설계도 포함된다. 1 -> 2 -> 3의 순서로 전산화가 진행되면서 살이 붙는 모습이다.

'데이터 모델링'에서의 '모델링'은 모델을 만들기 위한 과정이다. '모델'은 모델링의 대상들을 가시화, 문서화, 단순화시킨 결과다. 데이터 모델링의 결과물은 데이터 모델이다.

이 데이터 모델은 의사소통의 도구로 사용된다. 전산화에 필요한 모든 사람들이 데이터 모델링에 참여하여 같이 업무를 이해하는 것은 비용 낭비다. 그러므로 데이터 모델링에 참여하지 못한 사람들은 데이터 모델과 모델러와 현업의 상세한 설명을 통해 업무에 대한 이해한다. 여기에서도 갭이 발생하는데, 데이터 모델의 품질이 괜찮다면 구현 과정에서 갭이 좁혀진다.

데이터 모델링에 대해서 개략적으로 살펴보았다. 이번 절에서는 전산화가 업무의 이해를 필요로 하고, 전산화의 결과물이 현실과의 일치성이 결여되지 않도록 하는 것이 중요한 것임을 아는 것으로 충분하다. 데이터 모델링에 대해서는 많은 지면을 할애하여 설명할 것이다.

1.3 모델링의 우선순위(프로세스와 데이터)

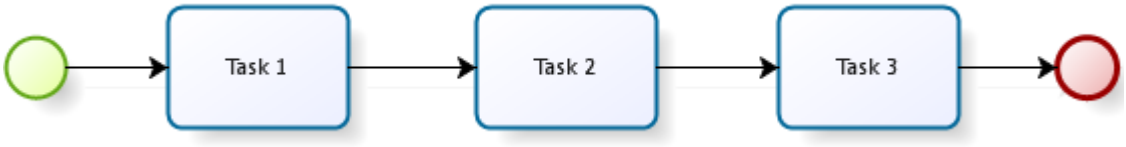
프로세스 모델링을 먼저 하느냐 데이터 모델링을 먼저 하느냐 아니면 동시에 진행하느냐는 오래된 논쟁거리다. 사람에 따라서 무엇이 우선하는지가 달라진다. 필자도 초기에는 데이터 모델링이 먼저 개발되어야 함을 주장했으나, 현재는 아무래도 상관없다는 입장이다. 왜냐하면 데이터나 프로세스는 현실에 그대로 존재하기 때문이다. 프로세스 모델을 먼저 만든다고 해서 데이터가 변하

[1]네이버 한자사전 참고

[2]계층형, 네트워크형, 객체지향형, 객체-관계형의 데이터 모델이 있으나, 객체-관계형이나관계형 데이터 모델이 대부분 사용된다.

는 것도 아니며, 데이터 모델을 먼저 만들고 나서야 프로세스 모델을 만든다고 해도 프로세스는 현실에 그대로 존재한다. 문제는 먼저 개발된 산출물이 이후 작업에 영향을 끼친다는 것이 문제인데, 이는 어떤 것이 먼저 개발되어야 하느냐의 문제가 아니라 산출물의 품질 문제다.

다음의 그림은 어떤 프로세스의 시작과 종료 사이의 여러 태스크들이 있음을 보여준다.



<그림 1.3.1>

Task1에서 필요한 데이터가 있을 것이고, Task2에서 필요한 데이터가 있을 것이다. 이렇게 각각의 태스크에서 필요한 데이터들이 모여 데이터 모델이 된다. 이러한 방식은 Bottom-Up 방식으로 각각의 개별 프로세스에는 잘 적합할 수 있으나 자칫 나무만 보고 숲을 보지 못하는 경우가 발생할 수 있으며, 데이터 모델의 품질은 전적으로 모델러의 역량에 달려있다. 여기서 필요한 모델러의 역량은 요구사항을 제대로 이끌어 내는 능력과 데이터를 통합하고 논리적으로 구조화하는 능력이다. 프로세스 모델을 먼저 개발하는 방법은 실무진들에게서 데이터 모델을 도출하는데 효과적이다.

반대로 데이터 모델을 먼저 개발하는 것은 Top-Down 방식으로 매우 어려운 일이다. 세상을 바라보는 관점 즉, 패러다임(관찰하는 방식 또는 틀)을 바꾸어 세상을 바라보고 추상화시켜야 한다. 관찰자(모델러)는 형이상학적 실재론이라는 이론을 적재하여 현실을 관찰해야 한다. 물론 처음부터 이론을 적재하여 관찰하는 것은 어려우므로 이 역시 연습이 필요하다. 이러한 연습이 없이 데이터 모델링을 했다면 반드시 그 대가를 치르게 될 것이다.

데이터 모델을 먼저 개발하는 방법은 실무진보다 넓은 관점을 가진 고위층들에게서 도출하는데 효과적이다. 어려운 점은 데이터의 쓰임새를 잘 판단할 수가 없다는데 있다. 그러므로 피드백 루프가 필요하다. 반드시 프로세스 모델이나 스토리 보드와 같은 문서를 보고 빠뜨린 곳이 없는지 확인해야 한다.

데이터 모델러는 현실을 관찰하고 구조화, 가시화시켜야 한다. 관찰을 위해서는 먼저 이론을 알고 있어야 한다. 이론은 다음 절에서 이어가도록 하자.

1.4 데이터 모델링 이론

사칙연산을 전산화 한다면 데이터 모델링은 필요치 않다. 이는 매우 단순하여 어떤 도구를 필요로 하지 않고 암산으로도 문제를 해결 할 수 있다. 하지만 우리가 대면한 프로젝트들은 이해할 수 있는 수준의 복잡성을 넘어선다. 우리의 선배들은 이러한 복잡성을 모델을 통해 단순화 할 수 있다는 것을 경험했으며, 이론을 정립하였다. 모델을 통해 요구 사항을 더 잘 이해할 수 있었고,

이는 프로젝트의 성공 가능성을 높이는 것임을 역사적으로 증명 하였다.

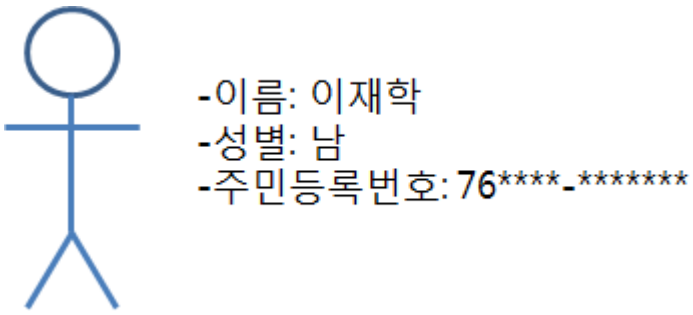
이번 절에서는 데이터 모델링에 근간이 되는 이론에 대해서 살펴볼 것인데, 다소 철학적인 내용이 포함되어 있다. 논리, 물리 모델은 이론적인 근거가 정확하다. 하지만, 개념 모델은 이론적인 근거가 무엇인지에 대한 내용이 기존의 문서들에는 자세히 소개되지 않고 있어 필자 나름대로 이론적 근거를 찾을 것이다. 철학적으로 논란이 되는 부분이 있을 것이지만, 이런 논쟁 따위는 쓰레기통에 처박아 버리자. 우리는 우리가 필요한 이론을 취하기만 하면 된다. 실무적인 내용들도 다소 포함될 것인데, 구체적인 실무에 대한 적용은 2장에서 정리 할 것이다.

컴퓨터 세계는 단순하지만 현실은 복잡하다

컴퓨터는 오로지 '1'과 '0'의 세계이다. 인터넷의 신문도 YouTube의 동영상도 '1'과 '0'의 조합이다. 컴퓨터 세계는 매우 단순하다. 그러나 현실은 어떠한가? 너무나도 복잡하다. 예를 들어 필자를 컴퓨터 세계에 표현한다고 해보자. 땀구멍의 개수나 위치, 주름, 지문 등을 모두 표현할 수 있을까? 외형적인 것뿐만 아니라 정신적인 것도 있으므로 100% 컴퓨터 세계로 옮기는 것은 불가능하다. 그러므로 현실 세계를 최대한 컴퓨터 세계로 반영시키되, 꼭 필요한 부분만을 반영해야 한다.

필요한 부분이 어떤 부분인지는 요구사항 조사/분석 과정에서 밝혀지기도 하고, 설계나 심지어는 구현된 제품을 인도하고 나서야 밝혀지기도 한다. 또한 똑같은 대상이라도 업무 영역에 따라서 요구사항이 다를 수 있다. 예를 들어, 인터넷 쇼핑몰에 계정을 생성 할 경우 이름, 주소 등을 입력하지만, 키나 몸무게와 같은 신체에 대한 정보는 기입하지 않는다. 결혼정보업체에 가입을 한다면 키와 몸무게는 필수 입력사항이 될 것이다. 경찰청과 같은 공공기관에서는 지문도 필요할 것이다. 그러므로 A프로젝트에서 사용했던 모델을 B프로젝트에 그대로 적용할 수는 없다. 물론, A, B 프로젝트가 같은 업종이거나, A프로젝트에서 사용했던 모델이 굉장한 고수준에서 일반화된 모델이라면 B프로젝트에 적용 할 수도 있다. 하지만, 추상화 수준이 낮다면 반드시 B프로젝트에 맞게 커스터마이징이 필요하다.

현실은 너무나도 복잡하고 표현하기가 어려워 현실적으로 컴퓨터 세계로 옮기려면 어느 정도 타협을 해야 한다. 이 타협을 '추상화(abstraction)'라고 한다. 다음의 그림을 보자.



<그림 1.4.1>

필자를 모델링(추상화) 해봤다. 필자를 굉장히 단순하게 표현했는데 어느 정도까지 표현해야 하는

지의 기준이 있어야 할 것이다. 그 기준은 단순하다. 필자인지 식별 가능한 만큼, 또 업무에 필요한 만큼만 표현하면 된다. 모델의 대상과 범위를 식별하고 업무에 어느 정도 자세하게 필요한지 알아내는 과정을 '모델링'이라고 하고, 결과물을 '모델'이라고 한다.

모델의 상세 수준을 결정하는 것을 '추상화 레벨의 결정'이라고 한다. 상세 수준은 요구 사항 분석을 통해 미래를 포함한 업무에 필요한 만큼이다. 위의 그림처럼 머리, 몸, 팔, 다리와 이름, 주민번호, 성별만 필요한지 아니면 눈, 코, 입도 필요한지 사용자의 요구사항을 수집/분석하여 추상화 레벨을 결정해야 한다. 상세하게 표현하면 할수록 추상화 레벨이 낮다고 말한다. 적절한 추상화 레벨의 선택은 시스템의 복잡성을 줄이고, 필요 없는 낭비 요소를 제거할 것이다.

개념은 매우 간단하다. 정보시스템을 구축할 때 추상화 레벨은 요구사항에 의해 결정되는데, 앞서 이야기 했듯이 100% 요구사항을 도출한다는 것은 신의 영역이므로 적절한 추상화 레벨을 결정하는 것은 매우 어려운 일이다. 추상화 레벨을 너무 낮추면 비효율이 발생할 수 있고, 추상화 레벨을 너무 높이면 사용자의 기대수준과 구현된 결과물과의 갭이 발생하게 된다. 이러한 갭 발생은 자연스럽게 사용자들의 외면으로 이어져 정보시스템의 생명주기(life cycle)가 짧아지게 된다. 그러므로 추상화 레벨이 높건 낮건 어느 한쪽으로 치우치면 반드시 비용낭비를 초래하므로 적당한 추상화 레벨을 찾도록 노력해야 한다.

참고:

> 추상화란 중요하지 않거나, 주 관심 대상이 아닌 자세한 부분은 감추거나 무시하고, 가장 중요하고, 근간이 되고, 다른 대상들과 구분될 수 있는 면만을 포함하고 있는 모델이며, 공통점을 강조하기 위해 차이점을 제거한 결과물 --객체기술 사전 (Firesmith, Eykholt, 1995) 에 정의된 추상화

관찰의 이론적재성과 데이터 모델링에 필요한 이론

일상적인 업무에서 문제가 발생했을 때에 체계적으로 접근하여 문제를 해결하는 사람이 있기도 하고, 시행착오법으로 해결하는 사람도 있다. 아는 만큼 보인다고 했다. 현상은 내 지식의 한계 속에 인식된다. 지식은 경험한-이론을 현실에 적용해 본-지식일 수도 있고, 경험하지 못했지만 이론적으로만 알고 있는 지식일수도 있다. 만약 어떤 현상을 관찰했을 때 왜 그런 현상이 발생하는지 알고 있다면, 그 현상을 관찰할 때 어떤 이론이 적재된 것이다. 이를 '관찰의 이론적재성'이라고 한다. 데이터 모델을 만들 때도 우선 겉으로 드러나 보이는 것을 관찰하게 되는데, 이때도 데이터 모델링과 관련된 이론이 적재되어야만 올바른 데이터 모델을 도출할 수 있다.

데이터 모델링은 주관적이라고 한다. 맞는 말이다. 왜냐하면 모델러가 여러 이론과 지식을 통합하여 현실 세계를 관찰하고 표현할 수 있는 수준이 다르기 때문이다. 만약 모델러가 현상에 대해 인식할 수 없다면 인식할 수 있는 수준으로 이끌어 내거나 학습을 해야 한다. 경험도 필요하다. 여기서 말하는 경험은 이론을 적용해 본 경험이다. 이론은 공통적인 것이지만 경험은 개별적인 것이다. 이론의 실무 적용 경험을 통해 더욱 성숙된 지식을 습득할 수 있으며, 이론 이외의 변수에 대한 고려도 함께 해볼 수 있다.

데이터 모델링에는 기본적으로 몇 가지 이론을 필요로 한다.

1. 형이상학적 실재론과 유명론[3]
2. 데이터 모델의 가시화 방법(데이터 모델을 표현하는 공통 언어)
3. 관계(relation)형 모델 이론

가시화 방법(표기법)은 일반적으로 ERD(Entity-Relationship Diagram)를 말한다. 형이상학적 실재론은 ERD에 표기되는 개체(entity), 관계(relationship), 속성(attribute)을 현실에서 관찰하고 파악하는데 필요한 이론이며, 관계(relation)형 모델 이론은 논리적인 관점에서 데이터를 표(table)형태로 관리하는지에 대한 이론이다.

표기법은 매우 쉽다. 간단한 심볼 몇 개만 익히면 되므로 이는 모델링 과정에서 자연스럽게 익숙해진다. 그러므로 데이터 모델링에 필요한 이론인 형이상학적 실재론을 먼저 살펴보고 릴레이션형 모델 이론을 살펴보도록 하자.

'관계(relation)'의 잘못된 번역[4]

앞서 언급했듯이 '관계(relation)형 모델'은 데이터를 표(table) 형태로 관리하고자 하는 모델이다. 여기서 말하는 '관계(relation)'는 'relationship'보다는 'table'에 유사하다고 할 수 있다. 보통 ERD에서 표현되는 '선분'이나 '마름모'가 아니란 얘기다. 한 가지 예로 필자가 정규화된 모델을 들고 개발자들과 이야기를 했더니만 'RDB스럽다'고 말하는 것이다. 정규화된 모델이건 정규화되지 않은 모델이건 'RDB'이지 'RDB'가 아닌 것은 아니다. 'relation'을 '관계'라고 번역한 것은 정말이지 최악의 번역이다. '관계'라고 번역한 순간 우리 후배들의 고생길을 열어준 것이다. 필자는 아예 관계(relation)는 표(table)를 의미한다고 기억하도록 하자는 의미에서 이후로는 '관계형'이란 용어 대신 이후부터는 '릴레이션형'이라고 부르도록 하겠다.

1.5 형이상학적 실재론

형이상학적 실재론은 존재하는 모든 것의 본성과 구조를 탐구하는 철학의 한 분과다. 대표적인 철학자로 플라톤, 아리스토텔레스, 칸트 등이 있는데, 그들은 각자 나름대로의 이론을 펼치고 있다. 또한 반대되는 이론(유명론)도 있다. 철학자들이 많은 논쟁을 벌이고 있지만, 그렇다고 여기에서 철학적인 논쟁을 파고들어갈 필요도 없고 필자의 능력도 없다. 필자는 논쟁 따위는 전혀 관심이 없다. 단지 데이터 모델링을 위한 이론이 필요했을 뿐이다. 우연인지 필연인지는 모르겠으나 형이상학적 실재론이 필자가 원하는 답을 해주고 있어 데이터 모델링은 근간이 되는 이론으로 삼았다.

[3]실재론이 대부분이지만, 유명론도 일부 필요하다.

[4]데이터베이스론 책이나 문서들에서 '관계형'이라는 단어를 모두 없애자는 캠페인이라도 해야겠다.

데이터 모델링은 속성(attribute), 관계(relationship), 종(kind)을 현실에서 찾아가는 과정이다. 형이상학적 실재론에서도 속성, 관계, 종에 대한 이야기를 한다. 개념적으로 봤을 때 데이터 모델링의 속성은 형이상학적 실재론의 속성과 일치한다. 관계도 일치한다. 일반적으로 개체 또는 개체타입이라 불리는 네모난 박스는 종과 일치한다. 1.5절에서는 형이상학적 실재론을 관점에 적재하여 세상을 관찰해 보는 연습을 해 볼 것이다. 그리고 이론을 설명할 것이다.

참고:

데이터 모델링에서는 '개체타입'을 그냥 줄여서 '개체'라고 표현하기도 하고, '개체집합(entity set)'이라고 하기도 한다. 그러나 무엇을 이야기하는지 잘 따져보면 이것들은 '종' 또는 '유' 중에 하나다. 아리스토텔레스는 제1실체(개체들)와 제2실체(종, 유)로 구분하고, 제1실체의 우선성을 강조했다. 데이터 모델링에서 Bottom-up 방식으로 모델링 할 때는 주로 제1실체의 관점에서 현실을 살피고, Top-Down 방식에서는 제2실체의 관점에서 현실을 살펴야 한다.

형이상학적 실재론의 관점으로 세상을 바라보자

그저 데이터 모델링의 배경이 되는 이론이 무엇인지 알고, 그것이 실무적으로 잘 활용만 될 수 있으면 된다. 철학적인 것은 경험상 주저리 주저리 떠드는 것보다 이론을 현실에 먼저 적용해 보는 것이 이해가 빠르다. 이론은 후에 설명할 것이다. 아래의 <그림 1.5.1>를 보자.

<그림 1.5.1>

독자는 이 그림에 보이는 여러 사물들을 관찰하며, 아래의 표에 분류를 해보라. 아래 표에는 4가

지까지만 분류하도록 해 놓았으나 필요하다면 더 분류해도 좋다. 하지만 이제 여러분의 세계에는 <그림 1.5.1>의 9가지 사물밖에 없다는 가정을 해야 한다. 분류를 하다 보면 애매모호한 경우가 있는데, 이는 명확해질 때까지 더 상세하게 분석해 보아야 함을 원칙으로 하면 된다.

분류한 명칭	분류한 사물

<표 1.3.1>

독자가 다 분류했으면, 필자가 분류해 나가면서 하나씩 살펴보도록 하자. 어떻게 분류하는지는 유사함으로 판단할 수 있는데, 유사함이란 속성(attribute)이나 현상이 일치함을 말한다. 또한 유한 대상들은 같은 대상과 상호작용(관계, relationship) 할 수 있음을 의미하며, 종(kind) 이 일치함을 말하기도 한다. 즉, 속성, 관계, 종이 일치함은 같은 범주로 분류될 수 있음을 말한다.

먼저 ①담배(This Plus), ⑥담배(Marlboro)을 살펴보자. ①담배(This Plus)과 ⑥담배(Marlboro)은 '담배(cigarette)'다 담뱃잎(식물, Tabaco)을 주 재료로 하여 만들어진 흡연 제품이다. 어찌 보면 ①담배(This Plus), ④지포라이터, ⑥담배(Marlboro)도 유사성이 있다. ①담배(This Plus), ④지포라이터, ⑥담배(Marlboro)은 '네모남'이라는 속성과 일치함을 보인다. 하지만 ④지포라이터의 속성을 보면 ①담배(This Plus)과 ⑥담배(Marlboro)의 속성은 많이 다른 것을 알 수 있다.

사물	속성
①담배(This Plus)	타르함량, 니코틴함량, 가격, 제조사
⑥담배(Marlboro)	타르함량, 니코틴함량, 가격, 제조사
④지포라이터	모델번호, 가격, 제조년도, 제조월, 생산지역, 연료

<표 1.3.2>

또한 종(kind)이 다르다. ①담배(This Plus)과 ⑥담배(Marlboro)은 '흡연을 위한 제품'이며, ④지포라이터는 '불을 붙이는 휴대 도구'로 ①담배(This Plus), ⑥담배(Marlboro)은 유사하지만, ④지포라이터는 ①담배(This Plus), ⑥담배(Marlboro)과 유사하지 않음을 알 수 있다. '네모남'이라는 속성이 일치하지만 관계나 종은 일치하지 않으므로 같은 범주에 놓기는 어렵울 뿐더러 '네모남'이라는 속성은 우리의 관심이 아니다.

④지포라이터, ⑤가스점화기, ⑦일회용라이터는 '점화도구'다. 즉, 종(kind)이 일치한다. 또한 속성도 일치한다. 하지만 ⑧중화가스레인지는 '가스레인지'라는 종으로 자체적으로 발화하지 못한다. 그러므로 ④지포라이터, ⑤가스점화기, ⑦일회용라이터와는 유사하지 않음을 알 수 있다. 또한 '불을 붙이는 휴대 도구' 로써 ④지포라이터, ⑦일회용라이터는 ①담배(This Plus), ⑥담배(Marlboro)과 관계를 가진다. ⑤가스점화기도 '불을 붙이는 도구'라고 볼 수 있으나 일반적으로는 휴대하고 다니지는 않는다. ⑤가스점화기는 ⑧중화가스레인지와 아주 강한 관계를 가진다. 물론 ④지포라이터, ⑦일회용라이터로도 ⑧중화가스레인지에 불을 붙일 수는 있으므로 관계를 가진다고 볼 수 있으나

⑤가스점화기 보다는 훨씬 약한 관계를 가진다. 그러므로 ④지포라이터, ⑦일회용라이터는 ⑤가스점화기와 같은 범주에 있다고 보기엔 곤란하다.

②국자, ③프라이팬, ⑨양은냄비는 '조리할 때 쓰는 도구'라는 같은 종에 속한다. 하지만 ③프라이팬, ⑨양은냄비는 ⑧중화가스레인지와 관계를 가지고, ②국자는 ⑨양은냄비와 관계를 가지므로 유사하다 볼 수 없다. 하지만 ②국자는 ⑨양은냄비와만 관계를 가지므로 따로 분류를 해야 한다. 그러므로 최종적으로 다음과 같이 분류되었다.

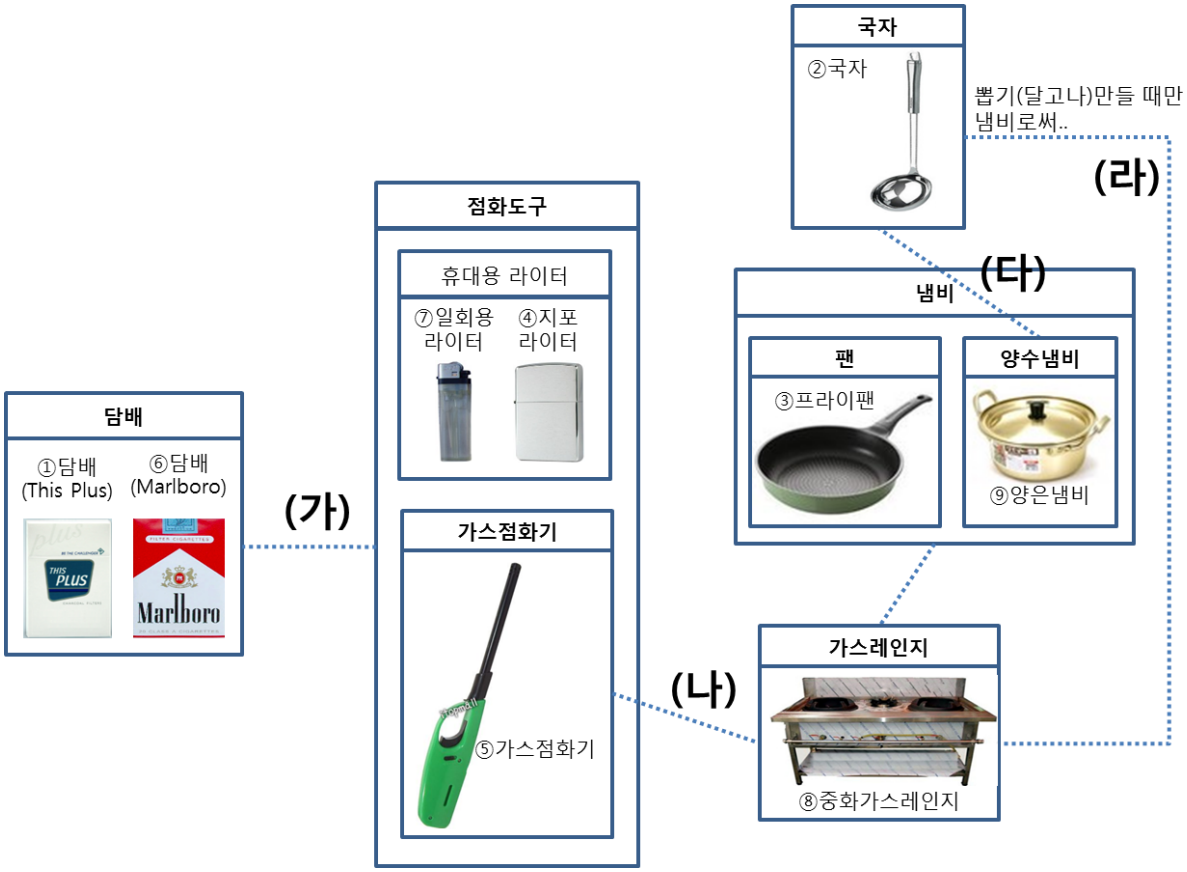
분류한 명칭	**분류한 사물**
담배(cigarette)	①담배(This Plus), ⑥담배(Marlboro)
라이터	④지포라이터, ⑦일회용라이터
가스레인지	⑧중화가스레인지
가스점화기	⑤가스점화기
국자	②국자
양수냄비	⑨양은냄비
팬	③프라이팬

<표 1.3.3>

이제 분류된 사물들을 가지고 사물들 간의 관계를 점선으로 표시한 관계도를 그려보면 <그림 1.5.2>과 같을 것이다. 주목해서 볼 부분은 필자가 표시한 (가), (나), (다), (라)이다. 점화도구와 냄비의 경우는 명확화를 위하여 범주를 상세하게 나눴다. 각각의 의미는 다음과 같다.

가) 담배는 점화도구로 불을 붙이다.
나) 가스점화기만 가스레인지에 불을 붙이는데 사용된다. 휴대용 라이터는 가스레인지와 관계 없음으로 규정한다.
다) 국자는 양은냄비에만 사용할 수 있다. 즉, 팬에는 사용될 수 없다.
라) 국자는 뽑기(달고나)를 만들 때만 냄비의 역할로써 가스레인지에 사용된다.

여기서 규정성의 차이가 발생한 부분을 보자. 대부분 담뱃불은 휴대용 라이터를 사용하지만 여기에서는 가스레인지에 사용되는 가스점화기로도 담뱃불을 붙일 수 있다고 규정했다. 국자는 양수냄비에서 국과 같은 것을 뜰 때에 사용되지만, 뽑기(달고나)를 만들 때만 냄비와 같은 역할을 할 수 있음을 규정했다. 그렇다면 국자는 냄비의 범주에 포함시킬 수 있지 않냐고 할 수 있다. 이것은 가스레인지의 관점에서만 냄비(type)의 범주에 속하는 것으로 국자와 양수냄비가 또 다른 관계를 가지므로 국자와 양수냄비를 다른 범주로 나누어 규성성의 차이를 확실이 보였다. 다른 대상들을 고려하지 않은 협소한 관점에서 분석은 지양해야 하고 전체적인 관점을 항상 유지하도록 해야 한다.



<그림 1.5.2>

이렇게 의미를 하나씩 살피다 보면 관계의 중요성을 알게 된다. 만약 점화도구(type)가 존재하지 않으면 흡연이 불가능 하므로 담배는 존재의 이유가 없어진다. 역으로 담배가 존재하지 않는다면 점화도구 중 휴대용 라이터는 존재의 이유가 없어지지만, 가스점화기는 가스레인지에 불을 붙여야 하므로 존재의 이유가 있다. (존재 종속)

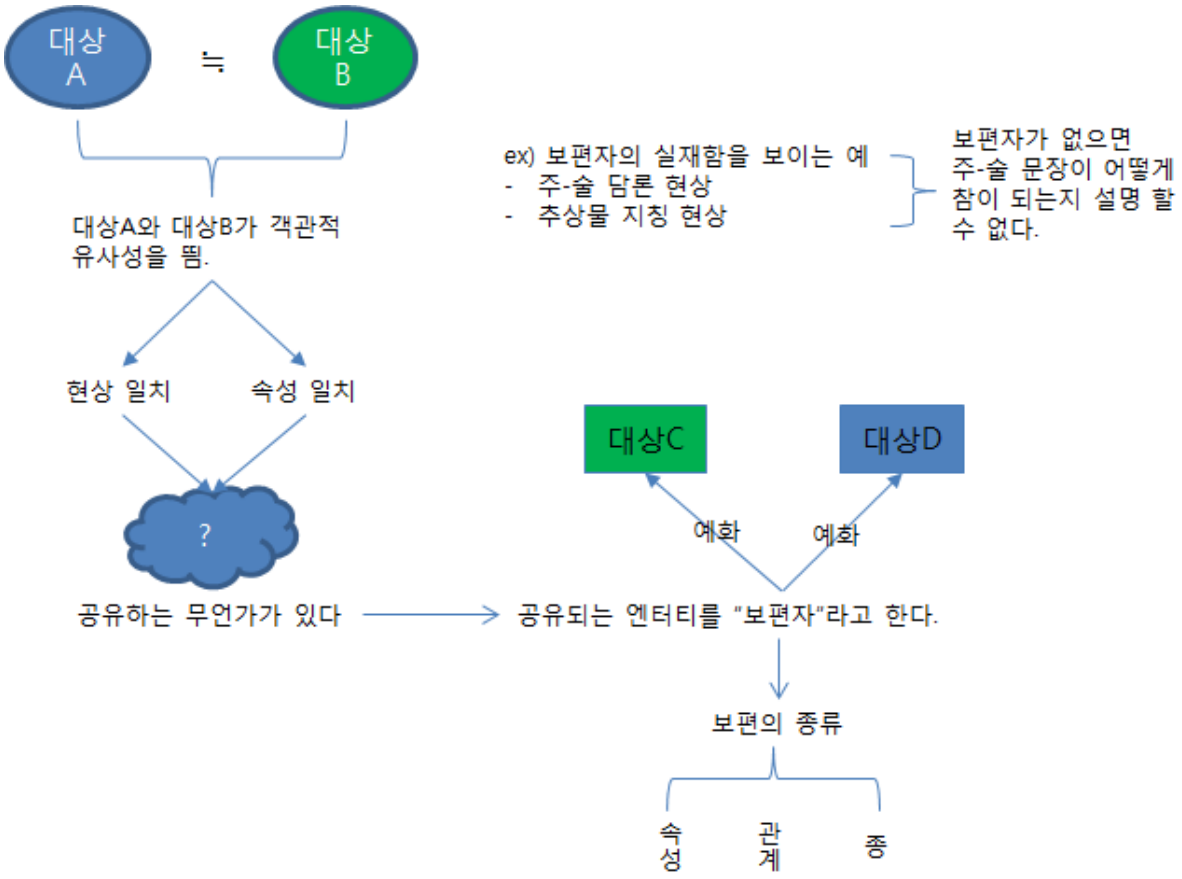
가스레인지는 냄비가 존재하거나 또는 국자가 존재하는 한 조리를 위한 불을 제공하는 역할로 존재의 이유가 있다. 하지만 냄비가 존재하지 않고 국자만 존재한다면 국자나 가스레인지의 존재가치는 냄비가 존재할 때보다 현저히 떨어질 것이다. 왜냐하면 국자나 가스레인지에 비해 냄비의 상호작용(관계)의 횟수가 더 많기 때문이다. 종과 유를 정리해보면 다음과 같다.

- 종: 담배, 휴대용 라이터, 가스점화기, 국자, 프라이팬, 양수냄비, 가스레인지
- 유: 점화도구, 냄비

이상으로 형이상학적 실재론의 관점에서 현실을 관찰해 보았다. 물론 철학적으로 논란이 되는 부분이 있지만 필자의 관점에서는 이런 논쟁은 불필요하다. 보편자(universal)의 존재여부, 종(kind), 유(genus)와 같은 정의적/추상적 보편자를 개체로 인정하는지 아니면 개체로 인정하는지는 중요치 않다. 플라톤인지 아리스토텔레스인지도 전혀 중요치 않다. 다만, 우리는 납득할 정도의 이론적 논리를 취하면 그만이다. 이제 이론을 살펴보자.

형이상학적 실재론– 이론

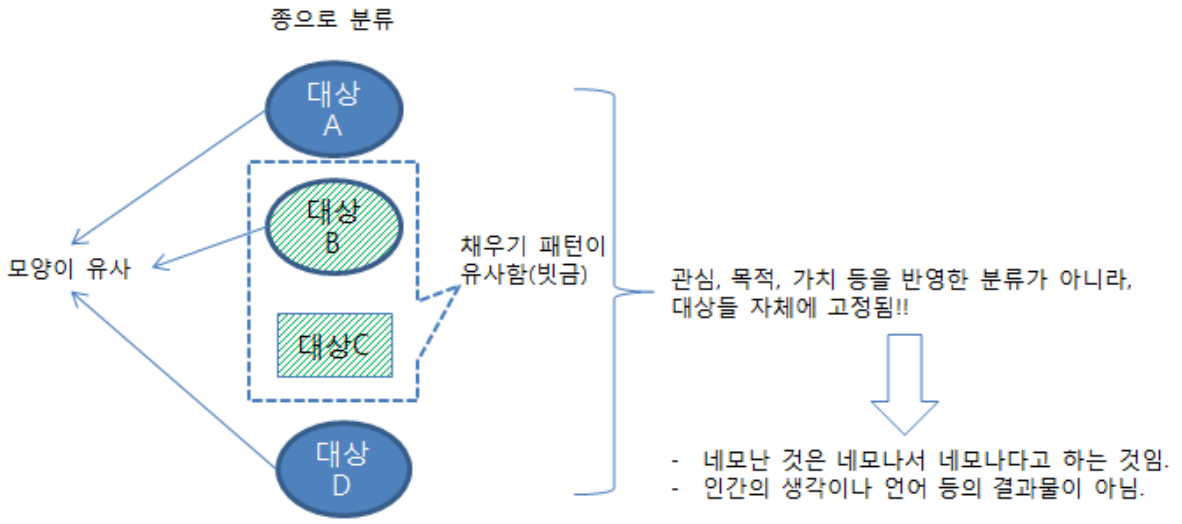
대상 A, B가 유사하다는 것은 현상 또는 속성이 일치함을 말한다. 속성이 일치함은 대상 A, B가 공유하는 무언가가 있음을 말한다. 여기서 대상 A, B가 공유하는 엔터티를 '보편자'[5]라고 한다. 여기서 대상 A, B는 예화(exemplified, 대상 A, B는 보편자의 instance)되었다고 말한다. 보편자는 속성(attribute), 관계(relationship), 종(kind)과 같은 것들이 있다.



<그림 1.5.3>

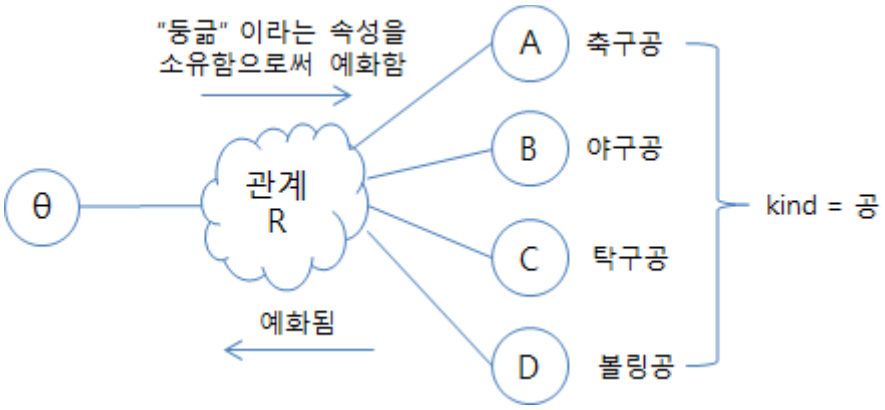
<그림 1.5.3>은 유사함에 대한 것으로 출발한다. 대상A와 대상B가 유사하다는 것은 현상과 속성이 일치함을 우리는 알고 있다. 유사함은 기본적으로 속성이 일치됨을 말하고, 속성이 일치된 대상들은 같은 무리로 분류될 수 있음을 말한다. 이러한 분류는 개인이나 조직의 관심, 목적, 가치 등을 반영한 분류가 아니라 대상들 자체에 고정된다.

[5]아리스토텔레스는 보편자를 개체(entity)로 인정하지 않는다.



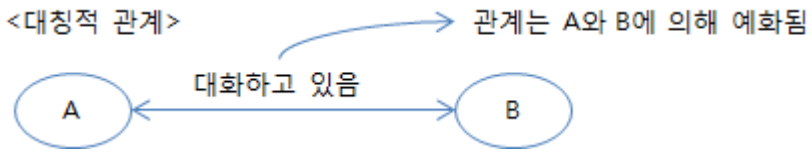
<그림 1.5.4>

보편자의 종류는 속성, 관계, 종이 있다고 했다. 예를 들어, 축구공, 야구공, 탁구공, 볼링공이 있다. 이 대상물들을 잘 생각해보자. 이 대상물들은 모두 둥글다. 이 대상물들은 '둥긂'이라는 속성을 소유함으로써 예화(exemplify)한다. 그리고 모두 공(ball)이라는 종에 속한다. 또한, 스포츠 중에 구기 종목들과 관계를 가진다.



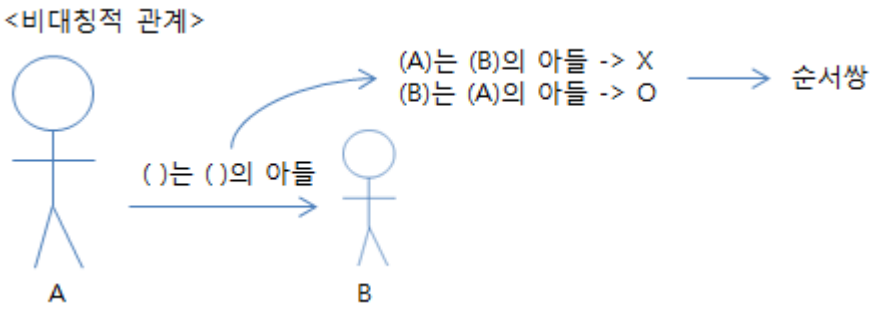
<그림 1.5.5>

관계는 대칭적 관계와 비대칭 관계가 있다. 대상A와 B가 대화를 하고 있다고 가정하자. 그러면 대상 A, B에 의해 "대화하고 있음"이라는 관계가 예화 된 것이다. 이는 대칭적 관계다.



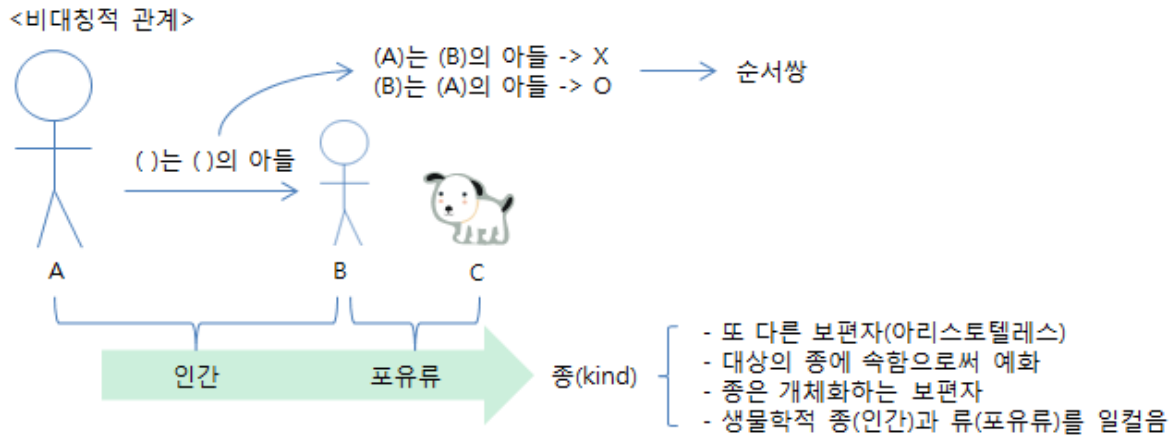
<그림 1.5.6>

비대칭적 관계의 대표적인 예는 아버지와 아들의 관계다. 사람 A, B가 있는데, 만약 A가 아버지라면 관계는 방향성을 가지게 된다.



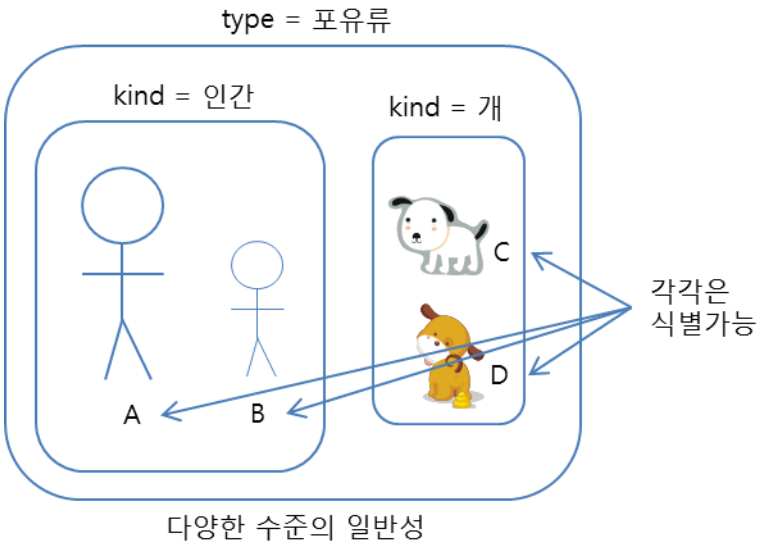
<그림 1.5.7>

대상 A가 아버지이고, 대상 B가 아들이면 A와 B는 '인간'이라는 종(kind)에 속한다. 대상 C도 있다. 대상 C는 강아지다. 그렇다면 A, B, C를 같은 종으로 분류하면 '포유류'로 분류할 수 있을 것이다.



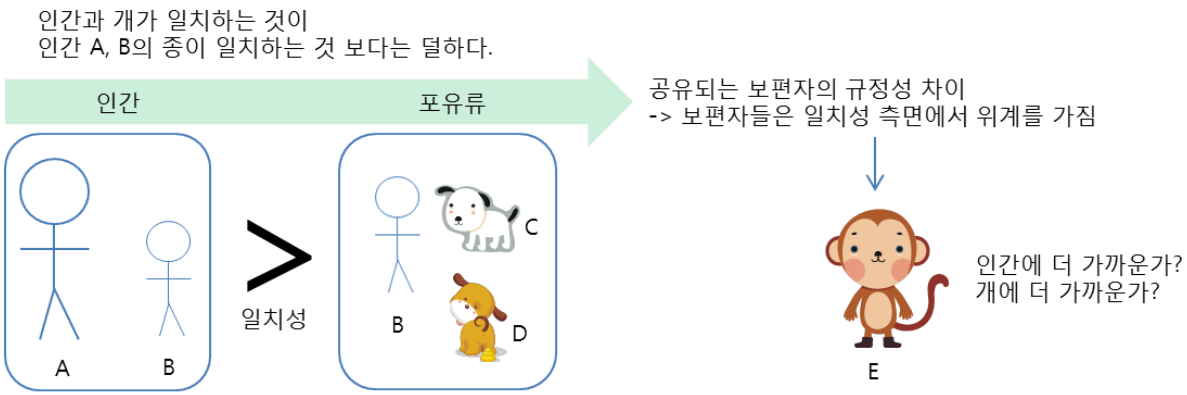
<그림 1.5.8>

위의 그림에서 인간 {A, B}로도 분류될 수 있고, 포유류 {A, B, C}로도 분류될 수 있음을 보았다. 이는 식별이 가능함과 종이 다양한 수준의 일반성을 가진다는 것을 뜻한다.



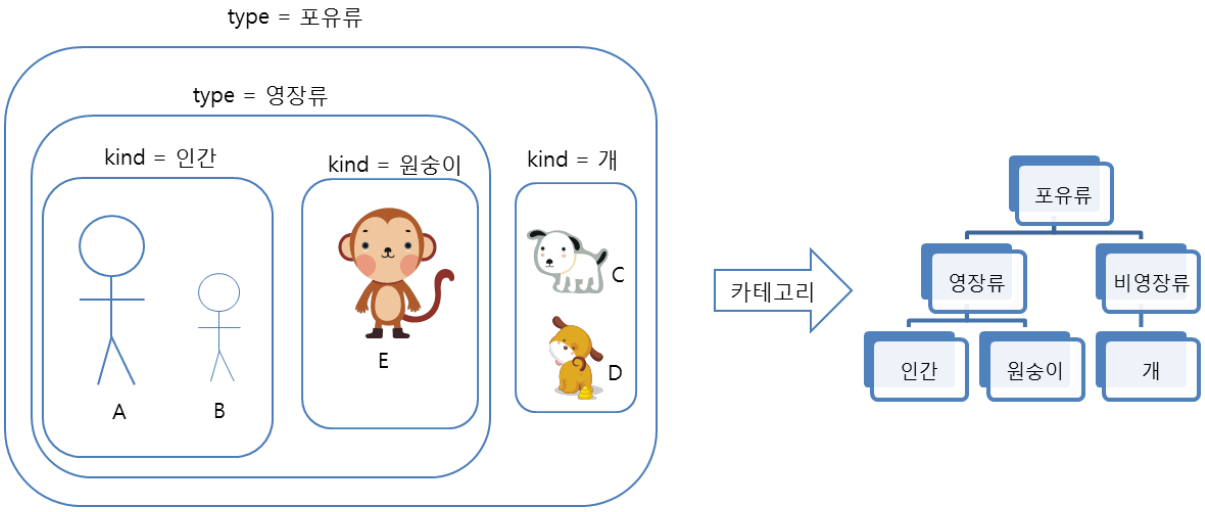
<그림 1.5.9>

실제로 대상 A, B는 매우 유사하다. B, C도 유사하다. 하지만 유사함의 정도는 B, C보다 A, B가 훨씬 유사함을 알 수 있다. 이는 보편자들 사이에 일치성 측면에서 위계가 있음을 뜻한다. 일반적으로 대분류, 중분류, 소분류와 같은 분류는 위계가 있는데, 위계의 기준은 일치성이다. 이렇게 어느 정도까지 규정할 것인지에 따라서 추상화 레벨이 결정된다.



<그림 1.5.10>

<그림 1.5.10>에서 대상 A, B, C, D, E는 포유류다. 그리고 대상 A, B는 인간이고, 대상 C, D는 개다. 그렇다면 대상 E는 인간에 더 가까운가 아니면 개에 더 가까운가? 물론, 인간에 더 가깝다. 그래서 다음과 같이 분류된다.



<그림 1.5.11>

이렇게 종과 유가 모이면 카테고리(category)가 만들어진다. 카테고리는 유의 개념으로 철학적으로는 더 이상 일반화 할 수 없다고 보여지는 유를 말한다. 데이터 모델링에서는 업무적으로 보았을 때에 최상위 유개념으로 보면 된다.

실제로 처음에 형이상학적 실재론을 응용했을 때 담배와 라이터가 등장했었다. 만약 담배와 라이

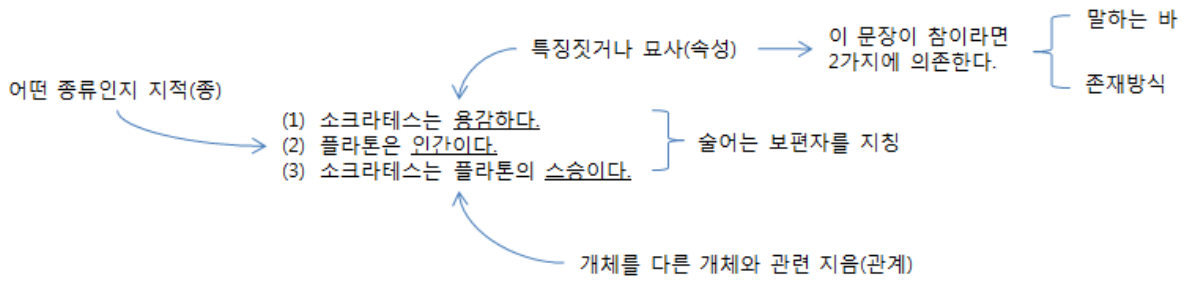
터가 담배가게에서 판매되는 상품이라면 다음과 같이 표현할 수 있을 것이다.

사물	속성
①담배(This Plus)	상품명, 타르함량, 니코틴함량, 가격, 제조사
⑥담배(Marlboro)	상품명, 타르함량, 니코틴함량, 가격, 제조사
④지포라이터	상품명, 모델번호, 가격, 제조년도, 제조월, 생산지역, 연료

<그림 1.5.12>

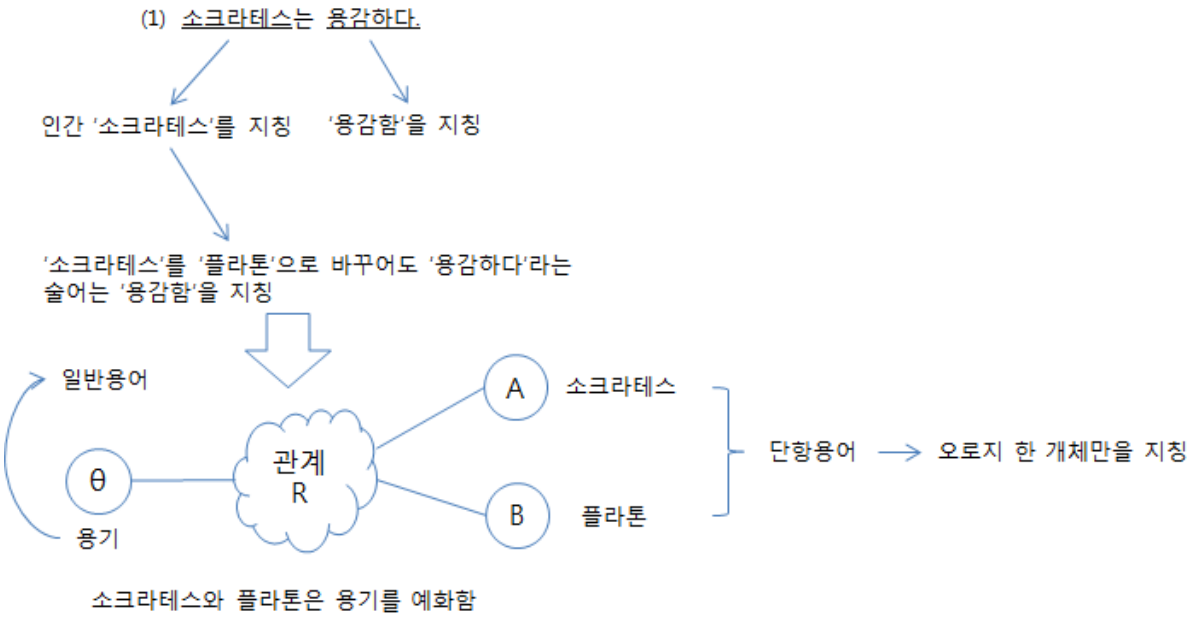
가격이라는 속성과 사물들이 관계를 맺고 있다. 단순히 상품이라고 분류할 수도 있겠지만, 좀 더 상세하게 관리하고자 한다면 상품은 담배와 라이터로 분류할 수 있을 것이다. 상세하게 분류하여 관리한다는 것은 담배와 라이터의 규정성의 다름을 의미하고, 다르게 관리하는 것을 의미한다.

형이상학적 실재론은 주-술 문장으로도 설명될 수 있다. 속성, 관계, 종을 다음의 3가지 문장에서 표현되는 것을 볼 수 있다.



<그림 1.5.13>

문장 (1)에서 '소크라테스'는 '단항 용어'다. 왜냐하면 세상에 '소크라테스'는 오호지 한 명뿐이기 때문(정체성)이다. 하지만 '소크라테스'를 '플라톤'으로 바꾸어도 '용감하다'라는 술어는 보편자 '용감함'을 여전히 지칭한다. 그래서 '용기'와 같은 단어는 '일반 용어'라고 한다. 일반 용어는 속성, 관계, 종 중에 하나임을 알 수 있다. 이를 그림으로 도식하면 다음과 같을 것이다.



<그림 1.5.14>

여기까지 데이터 모델링의 기본 이론이 되는 형이상학적 실재론을 정리하였다. 형이상학적 실재론이 존재한다고 주장하는 보편자가 실제로도 존재하느냐의 여부는 중요치 않다. 다만 우리는 이론이 제공하는 아이디어들을 이용하여 업무 영역을 이해를 높이고 고도로 구조화된 데이터베이스를 구축 할 수 있다면 그걸로 우리는 충분한 가치를 얻을 수 있다.

독자나 필자나 모두들 데이터베이스 구축 시 많은 노력을 기울여 속성, 관계, 종, 유형을 찾는다. 종은 아리스토텔레스에 이르러 보편자의 종류에 추가되었다. 데이터 모델링에서 종을 찾는 일이 가장 중요하고 가장 먼저 해야 하는 작업이다. 관계는 개체와 개체 사이의 상호작용이므로 종이 먼저 도출되어야만 관계가 도출될 수 있다. 속성은 규정성의 차이를 명확히 하여 종이 확실히 구별되어야 파악이 가능하다. 그러므로 역시 종이 먼저 도출되어야만 속성도 결정할 수 있다. 물론 개체들은 그저 속성들의 모임인 '다발'이나 '묶음'이라는 다발이론에 따르면 종이 먼저냐 속성이 먼저냐는 따지기 어렵다. 속성을 먼저 따져보는 방식은 상향식(bottom-up)이고, 종을 먼저 따져보는 것은 하향식(top-down)이다. 그러므로 상황에 따라서 적절한 방식을 취하면 되지만, 종을 먼저 알고 속성을 알면 속성이 가지는 의미를 명확히 알 수 있으므로 개체가 우선함을 원칙으로 하면 된다. 종을 도출하는 것은 매우 중요하므로 바로 밑에서 따로 떨어뜨려 설명하도록 하겠다.

종(kind) - 추상화(abstraction) 기법

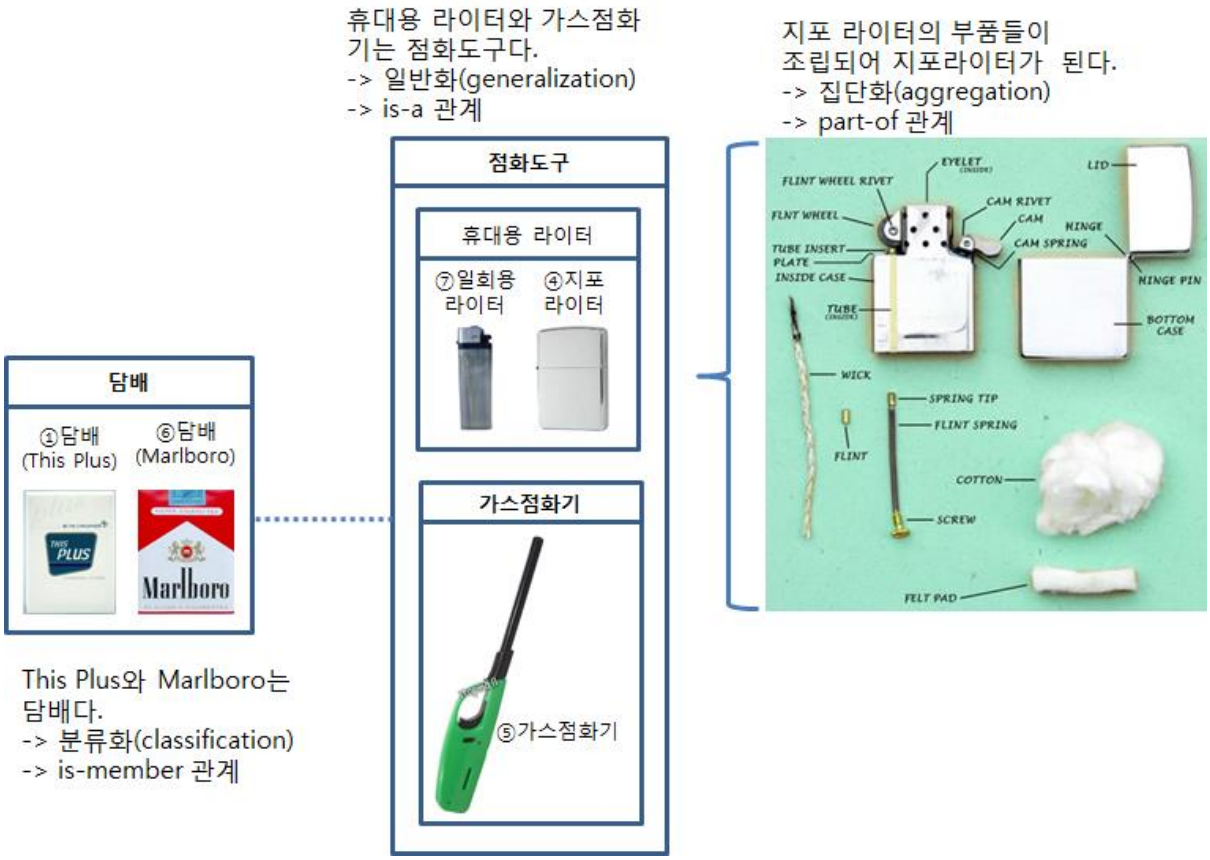
개개의 사물이나 개념 등을 보고 특정한 종을 구별하는 기본적인 방법은 아래의 3가지다. 이 기법들은 개념은 쉬운데, 많은 연습을 필요로 한다. 그러므로 여러분은 실생활에서 시간이 날 때마다 세상을 형이상학적으로 보도록 노력해야만 한다. 각각의 3가지 기법을 살펴보자.

기법	설명

분류화(classification)	공통의 성질에 의해서 특징지어짐
집단화(aggregation)	부분이 모여 하나가 됨
일반화(generalization)	종의 유형을 정의하는 것

<표 1.3.4>

3가지 방법 모두 상위 범주와 하위 범주와의 관계를 통해서 찾을 수 있다. 다음의 <그림 1.5.15>를 보면 쉽게 알 수 있다.



<그림 1.5.15>

역시 논란거리는 추상화 레벨의 결정(1.2절 참고)이다. 추상화 레벨은 요구사항에 의해 결정된다. 만약 라이터를 판매하는 선물가게라면 집단화 기법은 필요치 않았을 것이다. 왜냐하면 지포 라이터의 부품 하나하나를 관리할 필요가 없기 때문이다. 물론 부품을 모두 데이터베이스에 저장한다고 해도 이는 쓰이지 않는 데이터이므로 너무 낮은 추상화 벨은 비효율을 초래하게 된다. 하지만, 라이터를 생산하는 공장이라면 이러한 부품은 관리되어야 할 것이다.

중요한 것은 파악하고자 하는 대상이 무엇인지 알고 있어야 한다는 것이다. 파악하려는 대상은 사물(thing), 개념(concept), 사건(event), 행위(action) 중에 하나일 것이다. 형이상학적 실재론을 알아도 각각의 대상이나 개념이 인식되지 않으면 추상화될 수 없다. 그러므로 모델링의 대상이 되는 영역에 대한 지식이 있어야 한다. 지식이 없으면 현업 사용자에게 물어보거나 서적 등을 참고

해서 본질을 이해해야 하는데 시간을 많이 투자하여야 하므로 모델링 작업을 하기 전에 미리 학습이 되어 있는 것이 여러모로 좋다.

유(genus)와 종(species)

유(genus)와 종(species)은 '분류'를 한다는 데서 유사하며, 종(kind)으로 분류된다. 그렇다고 이음동의어도도 아니다. 왜냐하면 이 둘은 같은 종(kind)으로 분류되기 때문이다. 종은 유에 포함되는 관계다. 앞마당의 바둑이와 필자는 '포유류'다. 바둑이의 종은 '개'이고, 필자의 종은 '인간'이다. 유와 종은 상하관계가 있어 종에서 유의 방향으로 정의되면 될 수록 일반화(분류)되고, 종의 방향으로 정의되면 될수록 세분화(구분)된다. 즉, 종의 방향으로 정의되면 될수록 본질을 더욱 명확히 알 수 있다. 예를 들어, 앞마당에 뛰어 노는 바둑이가 일반적인 '개'에서 '포인터'라고 정의된다면 사냥개의 일종이며, 대형견이며, 생김새며 훨씬 더 명확해진다. 이렇게 따져보면 포유류-개-포인터와 같은 위계가 만들어졌음 볼 수 있다.

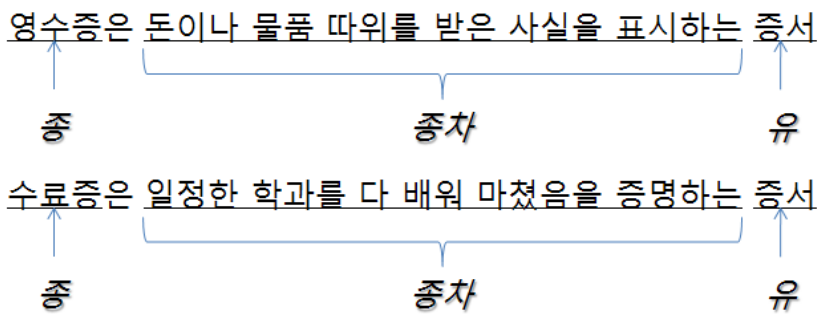
그렇다면 어디까지가 유이고 어디까지가 종인가? 그것은 상대적이다. '개'의 관점에서 보면 유는 '포유류'이며, '포인터'의 관점에서는 '개'가 유다. 이렇게 종 바로 위에 위치하고 있는 것을 '최근류'라고 한다.

하나의 유 아래에는 여러 종이 속 할 수 있다. 예를 들어, 포유류 밑에 개, 고양이, 고래가 위치한 것과 같다. 개, 고양이, 고래는 각각을 구별할 수 있는 차이점을 발견할 수 있는데, 이를 '종차'라고 한다.

유, 종, 종차는 어떤 대상을 정의할 때에 사용된다. 정의에는 정해진 형식과 규칙들이 있다. 예를 들어 다음의 정의를 보자.

- 영수증은 돈이나 물품 따위를 받은 사실을 표시하는 증서
- 수료증은 일정한 학과를 다 배워 마쳤음을 증명하는 증서

영수증과 수료증은 '종'이고, '증서'는 '유'를 나타낸다. 중간에 설명은 종의 차이를 설명하는 문장이다. 즉, 유, 종, 종차를 가지고 정의를 할 때는 "종 = 종차 + 유"의 과 같은 형식을 가져야 한다. 위의 설명을 그림으로 나타내자면 <그림 1.5.16>과 같다.



<그림 1.5.16>

유, 종, 종차를 가지고 정의할 때에도 여러 규칙을 지켜야 한다. 그 규칙은 다음과 같다.

규칙	잘못된 정의 예제
"종" ≒"종차 + 유"	개는 네발 달린 포유류이다. ('네발 달린 동물'은 개의 정의를 벗어남)
반복금지	개는 개다.
부정금지	개는 고양이가 아닌 포유류다.
명확한 진술	포인터는 작은 점이 많이 있는 큰 개다.
최근류의 사용	개는 식육목 개과의 생물이다.
종차의 특유속성 포함	개는 허파로 숨을 쉬는 포유류다.

<표 1.3.5>

참고:

이 문서에는 종을 이야기 할 때 생물학적인 유형의 종 뿐만 아니라 무형의 실체도 포함한다.

변화와 시간에 대한 문제

변하는 것은 보편자가 아닌 개체들이다. 개체들이 시간과 함께 지속한다는 것은 개체들의 변화와 시간이 관련이 있음을 뜻한다. 개체가 속성의 변화에 겪는 다음의 문장을 보자.

1. 고객번호가 1234 인 이재학은 2012-04-30 에 회원등급은 골드다.
2. 고객번호가 1234 인 이재학은 2012-05-01 에 회원등급은 실버다.

여기서 철학적인 논란거리는 2012-04-30의 이재학과 2012-05-01의 이재학이 같은 개체를 말하는지에 대한 것이다. 한 가지 견해는 1과 2 각각이 부분집합이고, 1과 2의 합집합이 하나의 개체라는 견해다. 다른 견해는 1과 2는 각 시간에 완전히 존재하는 독립적인 개체라는 견해다. 물론 2012-04-30의 이재학과 2012-05-01의 이재학은 같은 개체라는 것이 일반적인 견해다. 이 책에서도 이런 일반적인 견해를 취할 것이다.

이 책에서의 '시간'은 특수하게 다루어진다. 데이터베이스에 저장되는 데이터는 모두 같은 시계를 가지는 것을 대부분의 이해관계자들이 원한다. 왜냐하면 고객 A의 시계와 고객B의 시간이 다르다면 어떤 시간을 기준으로 하느냐에 따라 다른 값을 가지므로 데이터의 활용도가 떨어지기 때문이다. 그래서 일반적인 데이터베이스의 시간은 '전역적'이다. 전역적이라는 말은 데이터베이스 세계의 모든 개체들이 하나의 시간과 관계를 가지고, 보편자들은 항상 시간을 내포한 2차원이라는 의미다. 특별히 이야기 하지 않는 이상 개체들이 '현재'라는 시간 속성을 가지고 있음을 항상 인지해야 한다. 필자의 키가 187cm 라는 것은 항상 현재 또는 특정 시점이나 기간과 같은 어떤 시간적인 차원을 내포하고 있다는 것이다.

데이터베이스의 시간은 경우에 따라서 속성이 될 수도 있고, 개체로 취급될 수도 있다. 개체로 취급된다는 의미는 보편자가 존재함을 의미하게 된다. 운영계에서는 시간을 속성으로 취급하는 것이 보편적인 견해이며, 정보계는 시간을 개체로 취급하는 경향이 있다. 예를 들어 운영계에서는

입사일자, 가입일시와 같이 속성으로 취급된다. 또한 조직마다 그들만의 시간을 가는 경우도 있다. 창립기념일, 이벤트 기간, 휴일 등 시간이 조직마다 다르게 관리 된다. 그러다가 조직이 합쳐지면 2개의 시간을 가지게 된다. 경우에 따라서는 시간이 하나로 합쳐지게 되겠지만, 2개의 시간을 가져야 하는 합당한 이유가 있다면 그대로 2개의 시간이 존재하게 되고, 종(kind)이 만들어지게 된다. 즉, 보편자가 될 것이다.

데이터 모델링에서 변화에 대한 관리를 할 것인지에 대한 결정은 기본적으로 사용자의 요구사항에 달려있다. 물론 요구사항이 있다고 해도 자원(시간, 비용)이 충분치 않다면 자연스럽게 자원상황에 맞춰서 현재만 관리할 것인지 아니면 변화를 관리할 것인지 결정될 것이다. 한 가지 주의해야 할 점은 사용자가 요구사항을 말하지 않았다고 그냥 넘어가서는 안 된다는 것이다. 변화에 대한 관리를 기본으로 하여 상황에 맞게 현재만 관리해도 되는 것을 추려나가야 한다. 물론, 사용자들이야 '있으면 좋다'의 입장이거나 아니면 '현재는 가치판단이 어렵다' 정도의 입장을 취할 것이다. 이럴 때는 과감히 '없어도 무관한'으로 간주한다고 선언하는 것도 하나의 방법이다.

데이터 모델

데이터 모델은 다음과 같은 유형으로 분류된다. (모두 데이터에 관한 모델이고, 필요 이상으로 긴 명칭이므로 여기에서는 줄여서 말하도록 하겠다.)

- 개념 모델(=개념적 데이터 모델)
- 논리 모델(=논리적 데이터 모델)
- 물리 모델(=물리적 데이터 모델)

개념 모델은 개체-릴레이션형 모델(Entity-Relationship Model; ERD), 의미 객체 모델(Semantic Object Model)등의 유형이 있으나, 데이터베이스 분야에서는 누가 뭐랄 것도 없이 ERD가 업계 표준처럼 사용된다. 왜냐하면 ERD가 인간이 사유하는 방식-인간의 분류하여 기억한다-과 비슷한 구조를 가지고 있으며, 논리 모델인 릴레이션(relation)형 모델로 쉽게 변환되기 때문이다. 릴레이션형 모델은 데이터를 단순한 표(table) 형태로 관리하고자 하는 모델이므로 누구나 친숙하고 이해하기 쉬워 대부분의 프로젝트에서 채택된다. 또한 상용 DBMS제품들도 대부분RDBMS(Relational DataBase Management System)이므로 개념, 논리, 물리로의 자연스러운 흐름을 유지할 수 있다.

논리 모델은 릴레이션(relation)형 모델, 망(network) 모델, 계층형(hierarchy) 모델, 객체지향(object-oriented) 모델 등이 있다. 프로젝트 진행 시 망형 모델은 거의 찾아볼 수가 없으며, 계층형 모델의 경우 가끔씩 쓰인다. 계층 구조의 이점을 가져 물리적으로 구현했을 때에 검색 시 매우 빠른 성능을 나타내지만 변화에 취약한 계층구조를 유지해야 한다는 큰 단점이 있다. 그러므로 현재는 특정 분야를 빼고는 거의 사용되지 않는 모델이다. 앞서 언급했듯이 거의 대부분은 릴레이션형 모델을 사용한다. 또한 객체-릴레이션형 모델도 있는데, 객체지향형 모델과 릴레이션형 모델의 장점만을 수용한 모델이다. DBMS중 Oracle, PostgreSQL과 같은 DBMS가 이 모델을 구현하였다.

한 가지 아쉬운 점은 릴레이션형 모델이 집합이론과 1차 술어논리를 근간으로 하고 있음이다. 종

과 집합은 비슷해 보이지만 다르다. 집합은 원소들이나 명제가 어떤 집합인지를 결정하지만, 종은 그냥 그렇게 존재하고 있을 뿐이다. 예를 들어, 집합 S의 원소와 명제는 다음과 같다.

- S = {이재학, 고릴라G}
- 신장이 180cm 이상의 영장류들의 모임

여기서 실재론 관점에서 살펴보자. 집합 S의 원소 '이재학'은 '인간'이라는 종의 예화된 엔터티다. '고릴라G'는 '고릴라'라는 종이다. 인간과 고릴라로 구분되고, '영장류'로 분류된다. 여기서 인간 '김영임'은 집합 S에 속하는지는 명제에 철저히 따른다. 즉, '김영임'이 영장류로 분류되고, 신장이 180츠 이상이어야지만 집합 S에 속하게 된다.

여기서 주의할 점이 있다. 집합 S의 원소가 {이재학, 고릴라G} 라는 것뿐이지 집합 S가 영장류는 아니다. 즉, 집합과 종은 다른 것이다. 집합 S는 명제가 먼저 존재했을 수도 있고, 원소가 먼저 존재하여 명제를 결정했을 수도 있다. 하지만, 종은 개체들과 동시에 존재하거나 이미 그 이전에 존재한다. 개념 모델은 종 위주로 파악하지만, 논리 모델은 집합 개념을 사용한다. 이 차이를 모르면 애매모호한 부분이 생긴다. 이 문제를 해결하기 위해서는 빠르게 패러다임 전환을 할 수 있도록 연습하는 방법 밖에 없다.

물리 모델은 DBMS의 기능과 특성을 고려하여 테이블, 인덱스, 무결성, 뷰, 분산 등을 고려하여 데이터베이스 저장 구조로 변환하는 과정의 결과물을 말한다. 물리 모델을 생성하는 과정에서는 프로젝트의 비용, 인력, 사용패턴 등에 대한 고려도 필요하다. 특히, SQL과 DBMS 옵티마이저에 대한 전문적인 지식을 필요로 한다.

참고:

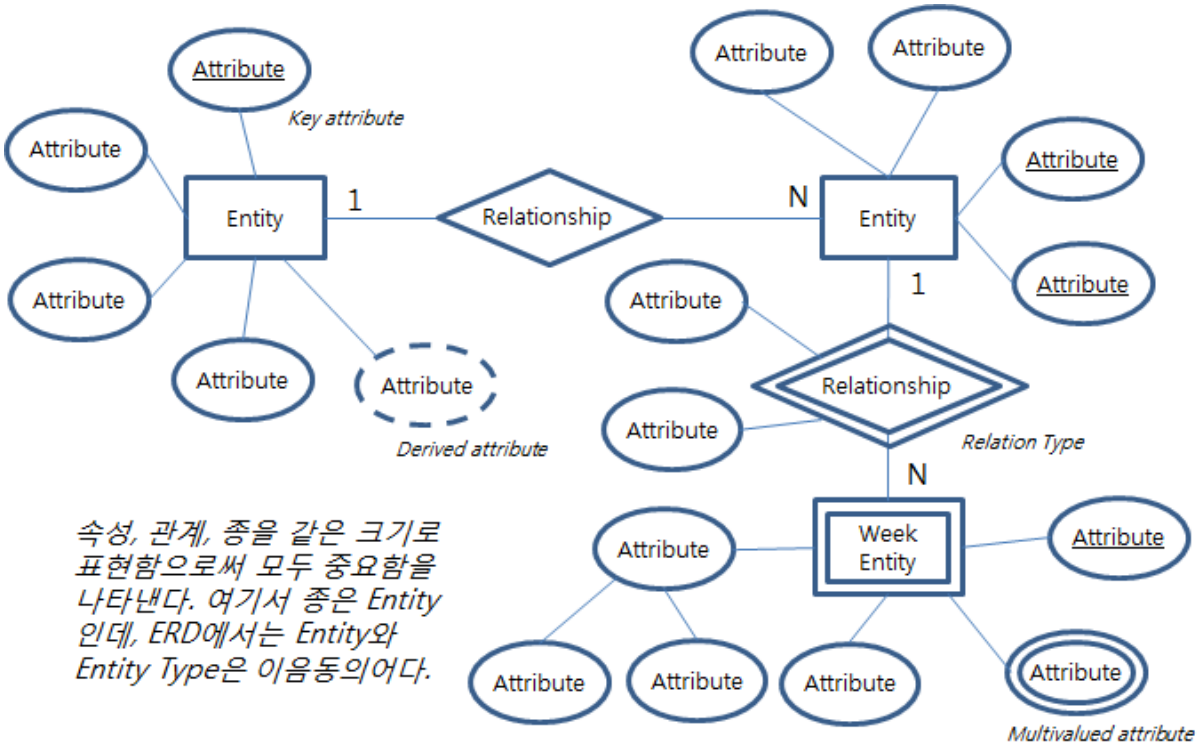
상용화된 데이터 모델링 툴이나 온라인 문서 등에서 개념 모델과, 논리 모델의 경계를 모호하게 표현하거나, 아예 개념 모델과 논리 모델은 이음동의어로 취급하기도 한다. 그 이유는 2가지다. 거의 대부분의 프로젝트에서 선택되는 논리 모델이 릴레이션(relation)형 모델이라는 것이 첫 번째 이유이고, 사실상 개념 모델을 만드는 과정을 논리 모델을 만드는 과정에 포함시켜도 된다는 것이 두 번째 이유다. 개념 모델과 논리 모델을 확실히 구분할 수 있는 한가지 방법은 사용되는 용어의 구분이다. 논리 모델에서 사용되는 용어를 모델을 설명할 때에 사용하는지를 판단하면 개념 모델인지 논리 모델인지 설명이 된다.

개념 모델

앞서 형이상적 실재론을 살펴보았다. 보편자의 종류로 속성, 관계, 종이 있다고 하였다. 그리고 이를 현실에서 도출도 해보았다. 여러분은 이미 개념 모델을 만드는 모든 이론과 실습을 모두 해본 것이다. 표준화된 방법으로 모델을 만드는 연습만 되면 된다. 이 절에서는 2개의 표기법을 소개할 것인데 속성, 관계, 종을 어떻게 표현되었는지에 대한 것만 살펴 볼 것이다. 유형은 is-a 관계로 표현하면 되므로 생략한다. 표기법 자체에 대한 설명은 부록에 있다..

앞서 <그림 1.5.2>과 같이 모델을 만들어 보았다. 아마도 우리의 선배들도 필자와 같은 생각에

기인하여 모델을 만들었을 것이다. 하지만 표준화된 표기법이 없어 일을 진행하는데 여러 가지 어려움이 있었을 것이다. 역시 이런 어려움을 없애고자 여러 표기법이 등장했다. 대학에서 기본적으로 사용하는 모델은Peter chen이 1976년에 제안한 모델로 표기법은 아래와 같다.



<그림 1.5.16>

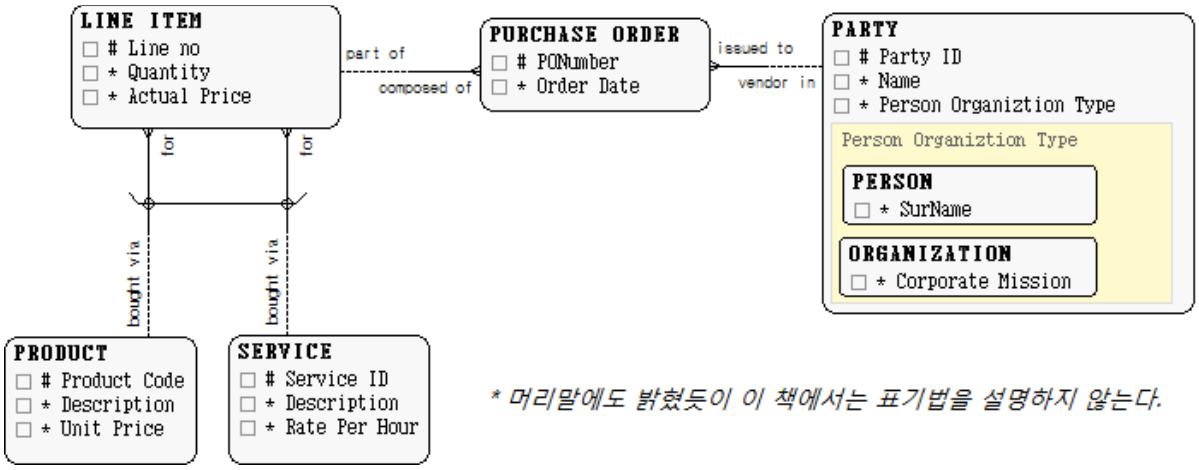
참고: 용어에 대하여..

위의 모델에서는 개체(Entity)는 종(kind)의 이음동의어다. 여러 사람마다 개체와 개체타입(Entity Type), 개체집합(Entity Sets)을 이음동의어로 취급하기도 한다. 동의어로 보는 것이 틀렸다고는 볼 수 없다. 플라톤의 실재론 관점에서는 보편자도 하나의 개체이니 말이다. 그러나 그냥 개체라고만 한다면 보편자의 예화 된 개체를 말하는 것인지 보편자 자체를 지칭하는 것인지 알 수 없으므로 이후부터는 개체, 종, 유형을 엄격히 구분할 것이다. 개체는 인스턴스(instance)와 같은 의미로 사용하도록 할 것이다. Peter chen의 모델을 설명할 때만 개체로 표기하겠다.

Peter chen 방식은 형이상학적 실재론과 매우 궁합이 잘 맞는다. Peter Chen의 의도인지 아닌지는 모르겠지만, 다른 모델들과는 다르게 기본적으로 속성, 관계, 개체의 심볼들이 모두 같은 크기로 표현되었다. 이는 속성, 관계, 종 중에서 어떤 것이 덜 중요하고 어떤 것이 더 중요하게 생각되는 것을 막아준다. 이는 형이상학적 실재론의 보편자인 속성, 관계, 종의 표현과 너무나 잘 맞는다. 하지만, <그림 1.5.16>에서도 보이듯이 심볼자체가 너무 크기 때문에 면적당 표현되는 정보가 적다는 치명적인 단점이 있다. 그래서 소규모 프로젝트라면 모를까 대부분의 프로젝트에서는 정보

공학, IDEF, Barker가 제안한 표기법을 많이 사용하는 편이다. 하지만 이들 표기법들은 논리 모델의 표기법에 가깝다.

필자가 즐겨 쓰는 표기법은 Barker 표기법이다. 데이터 모델링 툴로 ER-Win, Power Designer, DA# 등이 있는데, 필자가 즐겨 쓰는 데이터 모델링 툴인 DA#이 Barker표기법을 기본으로 한다. 예전에는 어떤 툴이던지 상관없었지만 현재는 DA#을 즐겨 쓰고 있다. 왜냐하면 데이터에 대한 표현력이 풍부하기 때문이다. 데이터 모델에 대한 표기법은 일관된 것이 좋으므로 이후의 모델은 모두 DA#을 이용한 Barker 표기법을 사용할 것이다. 아래는 Barker 표기법으로 표현한 모델이다.



<그림 1.5.17>

표기법이 바뀌어도 속성, 관계, 종, 유를 중심으로 표현된 것을 볼 수 있다. 이를 토대로 아래에서 설명할 릴레이션형 모델의 구조에 맞게 변형되어 결국은 논리 모델이 완성된다. 다음 절에서 릴레이션형 모델에 대해서 자세히 살펴보도록 하자.

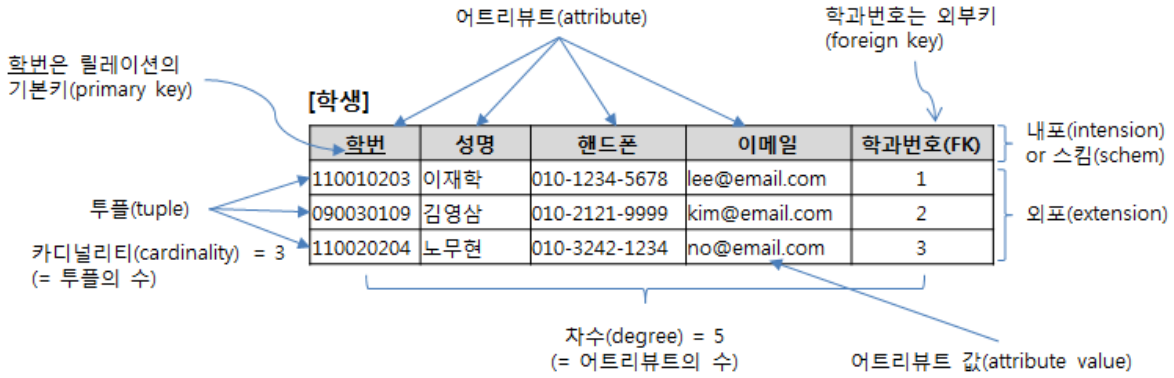
1.6 논리모델 – 릴레이션형 데이터 모델

릴레이션형 모델은 데이터를 표(table) 형태의 '릴레이션'으로 관리하고자 하는 모델이다. 릴레이션형 모델의 구성요소는 릴레이션, 릴레이션 대수와 해석, 제약조건이 있다. 릴레이션 대수와 해석은 실무에 쓰이지 않으므로 이 책에서는 SQL로 대체할 것이다.

릴레이션형 모델을 이해하려면 먼저 릴레이션의 구조와 명칭 알아야 한다. 초보자들이 가장 어렵다고 느끼는 부분은 용어인데, 이는 어려운 것이 아니라 익숙하지 않은 것이다. 그러므로 익숙해지도록 자주 보면 해결되는 문제다. 1.6절에서는 릴레이션의 구조를 먼저 살펴보고, 다른 구성요소에 대해서 설명하도록 하겠다. 정규화는 매우 중요하므로 1.7절에서 따로 다루도록 하겠다.

릴레이션의 기본 구조

'릴레이션(Relation)' 모델은1970년에 E.F Codd (1923 ~ 2003)가 데이터의 중복, 무결성, 독립성을 보장하기 위해 집합이론과 1차 술어논리를 기초로 하여 만들어진 데이터 모델이다. 릴레이션은 겉으로 보기에 2차원의 단순한 표(table) 형태이지만, 집합 이론과 1차 술어논리가 잘 스며들어 있는 데이터의 논리적인 구조다. 릴레이션이라는 명칭도 집합이론에서 따온 것이다. 각각의 어트리뷰트가 종속 관계에 의해 모여 <그림 1.6.1>과 같은 2차원 형태가 된다.



<그림 1.6.1>

릴레이션이 의미하는 바를 알기 위해서는 논리적인 기술(description, 또는 설명)이 필요하다. 이러한 논리적인 기술을 릴레이션 스킴(Schem) 또는 내포(Intension)라고 하며, 다음과 같은 형식으로 기술한다.

학생(학번, 이름, 핸드폰, 이메일)

밑줄(_)은 릴레이션의 기본키(primary key)를 의미하는데, 기본키는 각각의 투플을 식별하는데 사용되는 1 개의 어트리뷰트 또는 2개 이상의 어트리뷰트들이다. 투플은 개체나 속성 또는 관계의 인스턴스다. 즉, 릴레이션에서 어트리뷰트와 관계되는 값의 집합을 말한다. 위의 그림에서는 {한 명의 학생 = 투플}은 다음과 같다.

1234, 홍길동, 010-1234-3287, hong@email.com, 1

이런 투플들의 모임을 릴레이션 외포(extension)이라고 한다. 위의 <그림 1.6.1>에서 Student 릴레이션은 5개의 어트리뷰트를 가지는데 이는

차수(degree)가 5 이다

라고 표현한다. 5개의 어트리뷰트 중에 '학과번호'는 학과 릴레이션으로부터 즉, 외부에서 온 어트리뷰트로 이는 외부키다. 또한, 투플이 3개인데 이는

카디널리티(cardinality)가 3 다.

라고 표현한다.

학생을 추상화에서 대표적인 속성들을 추려내어 어트리뷰트 집합을 만든 것이 <그림 1.6.1>라고 가정해 보자. 여기서 대표적이라는 말은 추려진 어트리뷰트들(학번, 이름, 핸드폰, 이메일)이 있어야만 학생을 표현하고, 학생과 관련된 업무를 처리하는데 부족함이 없다는 것을 의미한다. 만약 대리출석을 막기 위하여 전자출석부에 학생의 사진이 필요하다는 요구사항 더 있었다면 <그림 1.6.1>의 학생 릴레이션에는 '사진'과 같은 비구조적인 어트리뷰트를 추가해야 할 것이다. 이처럼 릴레이션은 모든 사용자의 관점을 통합해서 표현해야 한다. 모든 사용자의 관점에서 릴레이션이 잘 만들어졌는지를 알아보는 대표적인 도구가 CRUD(Create, Read, Update, Delete) 매트릭스인데, 이것은 나중에 자세히 살펴볼 것이다.

참고:

이미 두 번째 언급하지만, 'Relation'을 '관계'로 번역한 것은 결과적으로 선배들의 명백한 과실이다. 필자가 또 이야기 하는 것은 잘못된 번역으로 인해 너무 많은 손해를 보고 있기 때문이다. 주변에 데이터베이스와 관련된 사람에게 물어보라. 우리가 흔히 이야기하는 관계형 데이터베이스에서 '관계'가 무엇을 의미하는지를. 정말 큰일이 아닐 수 없다. 이건 단순히 기술적인 문제가 아닌 패러다임의 문제다. 흔히 오해하고 있는 '관계(relationship)'의 패러다임은 개체간의 상호작용이나 개체간의 어떤 상태에 관한 것이다. 하지만, '릴레이션(relation)'의 패러다임은 데이터를 표(table) 형태로 관리하고자 하는 패러다임이다.

릴레이션의 특성 - 지켜져야만 진정한 릴레이션!!

릴레이션형 데이터베이스는 아래에 열거한 특성으로 인해 논리적인 독립성을 유지하고 파일 시스템에 비해 변화에 빨리 적응되도록 발전되어 왔다. 릴레이션형 모델에서 릴레이션은 다음과 같은 특성을 가짐으로써 다른 논리 모델과 차별화된다. 이러한 특성들로 인해 릴레이션형 모델은 변화에 매우 강한 구조를 가지게 된다.

- 투플의 유일성
- 투플들의 무순서
- 어트리뷰트들의 무순서
- 어트리뷰트의 원자값 --> 가장 이해하기 어렵고 중요한 특성이다.

투플의 유일성은 각각의 투플을 식별할 수 있어야 함을 의미한다. 차수가 5인 릴레이션에서 적어도 어트리뷰트 5개의 값의 조합이 유일(unique)해야 함을 의한다. 물론 5개가 아닌 2개, 1개의 어트리뷰트로도 각각의 투플을 식별할 수 있어도 되는데, 이러한 어트리뷰트나 어트리뷰트의 조합들을 슈퍼키(super key)라고 하며, 이들 중에서 대표가 될 만한 것들로 추려진 슈퍼키들을 후보키(candidate key)라고 하며, 후보키들 중에 각각의 투플들을 식별하는데 대표적인 의미를 가져서 선정된 어트리뷰트(들)이 기본키(primary key)가 된다. 만약 필요에 의해 DB설계자가 특정 목적으로 원래의 기본키 대신 다른 임의의 어트리뷰트를 추가하여 기본키 역할을 시키면 대체키(alternate key)가 된다. 대체키가 아닌 기본키를 자연키(natural key)라고 한다.

투플의 무순서는 다른 논리 모델과는 다른 릴레이션형 모델만의 특성이다. 만약 투플의 순서에 의미를 부여하고 싶다면 정렬 연산을 하거나 따로 순서에 대한 값을 갖는 어트리뷰트를 두어야 한다. 위의 <그림 1.6.1>에서 '홍길동'이 '이순신'보다 먼저 존재한다고 해도 먼저 입학하거나, 성적이 우수하거나 나이가 많거나 하는 어떤 가치판단의 기준이 되는 것이 아니다. 그러한 가치판단은 어트리뷰트 값으로 결정한다. 예를 들어, '입학일자'라는 어트리뷰트를 오름차순으로 정렬했을 때 투플의 순서는 입학을 먼저 했는지 나중에 했는지의 가치판단을 할 수 있게 된다. 릴레이션이 물리적으로 구현될 때에 입력의 순서와 상관없이 create index, table rebuild와 같은 작업에 의해 순서가 변경될 수 있다. 그러므로 투플의 순서에 어떤 의미를 부여했다면 릴레이션형 모델의 사상에 반하는 것이 된다. 또한 투플의 순서에 의미를 둔다는 것은 데이터베이스의 장점인 물리적 독립성을 해치는 것이다.

어트리뷰트들의 무순서도 논리적인 독립성을 유지하는데 필수적인 역할을 하는 특성이다. 어트리뷰트의 무순서는 릴레이션에서 필요한 어트리뷰트를 사용자로 하여금 취사 선택하도록 강제하고 있다. 릴레이션형 모델은 3단계 스키마 구조의 '개념적 스키마'에 해당되므로 모든 관점을 통합해 놓았기 때문에 어트리뷰트의 순서에 의미를 부여한다면 릴레이션형 모델은 변화에 취약하고, 최소한의 중복을 지향할 수 없게 되어 결국은 견고하지 못한 모델이 된다. 그러므로 실제로 SQL에서 'select * from tableA'와 같이 '*'의 사용은 릴레이션형 모델의 특성에 반하는 것으로 종속성을 가지게 되어 유지보수 비용을 증가시키게 된다.

어트리뷰트의 원자값은 어트리뷰트 값이 하나의 값과 하나의 의미만을 가지는 것을 뜻한다. 원자값이라고 해서 무조건 쪼개지지 않는 단위까지 쪼개는 것은 아니다. 예를 들어, '성명'을'성', '명'으로 쪼개는 것이 반드시 원자값으로 쪼갰다고는 할 수 없다. 원자값의 원자 단위는 업무 처리가 어떻게 이루어지느냐에 따라서 다르다. 우리나라의 경우는 성과 이름을 같이 붙여서 사용하므로 성명이 원자값이지만, 외국의 경우는 결혼을 하면 성이 변경되어 사용되는 경우가 있기 때문에 성, 명을 분리하는 것이 원자값이 된다. 이는 데이터 모델링이나 DB설계 단계에서 결정되는 매우 중요한 특성이다. 원자값에 대한 고려를 하지 않는다면 DB를 사용하는 사용자(사람, 정보시스템..)가 고생을 하게 된다. 이는 비용증가다.

어트리뷰트-도메인, 설계어트리뷰트, 유도어트리뷰트

각각의 어트리뷰트는 도메인(domain)을 가진다. 도메인은 어트리뷰트가 가질 수 있는 값의 명세를 말한다. 예를 들어, 성별이라면 문자형의 '남', '여'의 값만 가질 수 있으며, 'null 값을 가지지 못한다'와 같이 어트리뷰트의 도메인을 정의할 수 있다. 도메인은 DBMS에 종속적인 것이 아닌 범용적인 정의이다. 이는 마치 도메인을 정의하는 모습은 보편자인 속성이 여러 개체들과 관계를 가진 모양이다. 아래는 학생.학생번호 어트리뷰트의 도메인의 예제다.

이름(한글)	학생번호
이름(영문)	Student Number
약어	SNo

의미	학생 릴레이션에서 각각의 학생을 구분 지을 수 있는 설계어트리뷰트
데이터 타입	Char
길이	9
형식	[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]AND 학생번호 > 0
숫자정확도	
허용범위 (입력 가능한 값의 목록)	000000000 ~ 999999999
유일성	Unique
널값허용	Not Null
기본값	
룰	- 1~2번째 자리: YY 년도 - 3~5번째 자리: 학과번호 - 6 ~9번째 자리: 일련번호

<표 1.6.1>

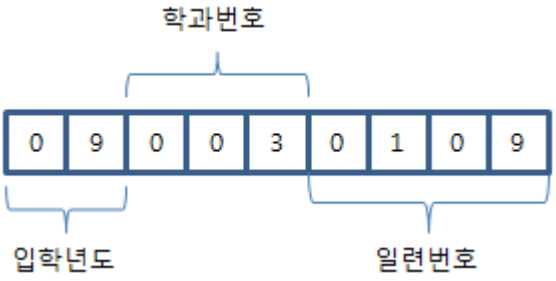
학생번호와 같이 다른 어트리뷰트에 의해 계산된 어트리뷰트를 유도(Derived) 어트리뷰트라고 한다. 유도어트리뷰트는 어플리케이션에 의해 구현되기도 하고, DBMS에서 명시적으로 관리되기도 한다. 문제는 어플리케이션에서 유도 규칙이 구현된 경우인데, 무엇이 문제인지는 데이터베이스의 정의를 통해 드러난다. 아래는 데이터베이스의 정의다.

> *특정 조직 내에서 다수의 사용자들이 공유(Share)할 수 있도록 통합(integrate)시키고 컴퓨터 저장 장치에 저장(Store)시킨 운영(operation) 데이터의 집합이다.*
>
> *--이석호, 데이터베이스 시스템, 정익사*

여기에서 '다수의 사용자들이 공유(Share)'라는 말에 주목하자. 여기서 사용자는 사람을 의미 할 수도 있고, 어플리케이션을 의미할 수도 있다. 만약 A, B, C라는 사용자가 있고, A가 유도 규칙이 구현된 어플리케이션이라 가정하자. 만약 B와 C도 유도 규칙을 구현하지 않는다면 어트리뷰트의 도메인을 벗어날 가능성을 내포하고 있는 것이다. 데이터 거버넌스(Governance)가 있거나 DBA에 의해 관리가 가능하다면 조율이 가능하겠지만, Ad-Hoc Query의 경우는 관리 불가능하므로 데이터 품질 저하의 가능성은 항상 내포하고 있는 것이다. 이렇게 어트리뷰트의 도메인에서 어트리뷰트 값이 벗어나는지에 대한 제약조건을 '도메인 제약조건 (domain constraints)'라고 부른다.

설계(designed) 어트리뷰트는 관리나 개발에 편의성을 더하기 위해 추가되는 어트리뷰트다. 논리 모델에서 생성되기 보다는 물리적인 구현 시 추가되는 경우가 많다. 왜냐하면 그때서야 알려지지 않은 관리 패턴이나 사용 패턴이 드러나기 때문이다. 예를 들면, 데이터를 바로 삭제하지 않고 '삭제여부'와 같은 어트리뷰트를 두어 비동기적으로 배치(batch)처리하는 경우가 설계 어트리뷰트를 사용한 경우다. 또한 데이터를 실제로 삭제하기 전에 유예기간을 두어야 한다는 업무 제약조건이 있는 경우는 '삭제일시'라는 어트리뷰트를 두어 업무를 지원하는 경우도 설계어트리뷰트의 한 예로 볼 수 있다.

<표 1.6.1>에서 학생번호 어트리뷰트는 문제점이 있음을 미리 말했다. 이 어트리뷰트의 문제는 릴레이션의 특성인 '어트리뷰트의 원자값'이라는 대목에서 문제가 된다. 예를 들어, 학번은 다음과 같이 될 것이다.



<그림 1.6.2>

입학년도, 학과번호, 일련번호라는 코드체계를 가진 학생번호는 원자값이 아니다. 더욱이 학생번호는 항상 입학년도, 학과번호에 종속적이다. 그러므로 관리포인트가 늘어난다. 릴레이션이 다음과 같이 꾸며졌다면 어떨까?

[학과]

학과번호	학과명
1	컴퓨터공학과
2	전자공학과
3	정보통신공학과
4	건축공학과

[학생]

학번	성명	핸드폰	이메일	입학년도	학과번호	일련번호
1	이재학	010-1234-5678	lee@email.com	2011	1	203
2	김영삼	010-2121-9999	kim@email.com	2009	3	109
3	노무현	010-3242-1234	no@email.com	2011	2	204
4	김대중	010-1000-2000	kim2@email.com	2007	2	111
5	전두환	010-0505-5555	jdh@email.com	2008	4	9
6	이승만	010-0101-1111	seng@email.com	2010	4	5

<그림 1.6.3>

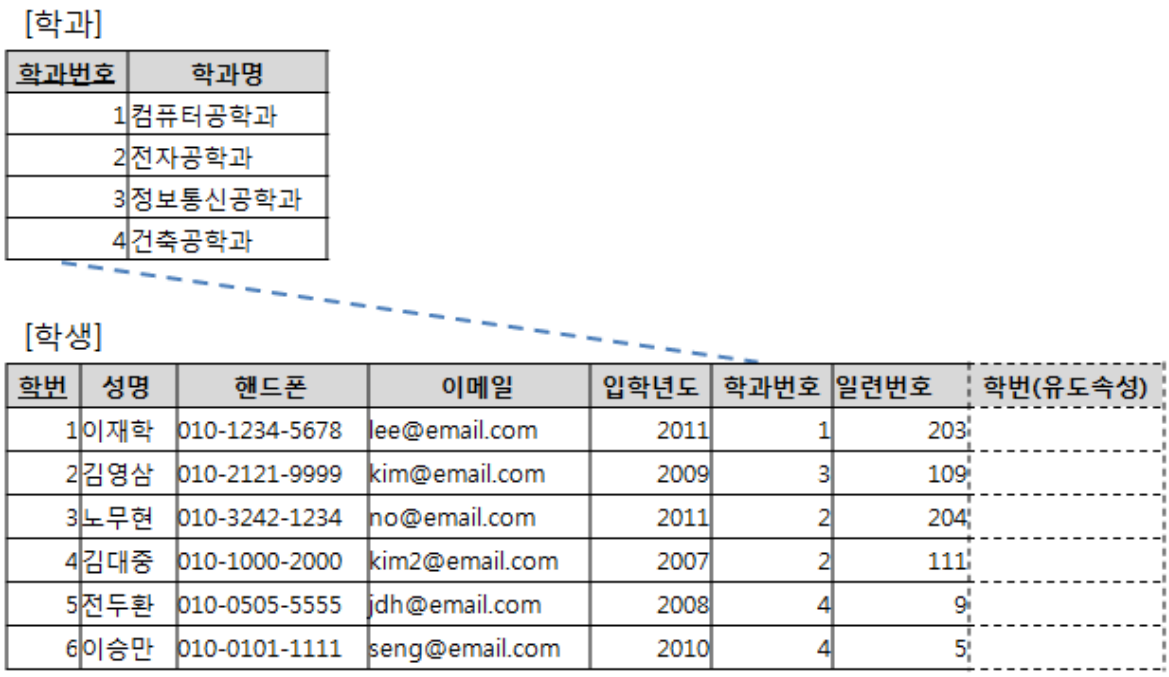
<그림 1.6.3>의 학생 릴레이션에는 어딜 봐도 'YYDDDXXXX' 형식의 학생번호는 없다. 물론 입학년도, 학과번호, 일련번호를 적절히 조합하면 9자리의 학생번호를 만들어 낼 수는 있다. 아래와 같이 SQL을 작성하여 사용할 수도 있다.

```
DECLARE
        @학번char(9)
```

```
SET@학번='090030109'

SELECT성명,핸드폰,이메일
FROM학생
WHERE입학년도='20'+LEFT(@학번,2)
AND학과번호=SUBSTRING(@학번, 3,3)
AND일련번호=RIGHT(@학번, 4)
```

문제는 타성이나 관습에 대한 도전이다. 교수님들이 출석을 부를 때나, 학사행정 등의 학생과 관련된 일이라면 이제까지는 9자리 학번을 사용한 것이 이제까지의 관습이며 역사다. SQL을 작성할 때도 "WHERE 학번 LIKE '11003%' "와 같은 조건으로 2011년에 입학한 정보통신공학과 학생들을 쉽게 찾을 수 있었지만, 이제는 "WHERE입학년도 = 2011 AND 학과번호 = 3'과 같이 입력해야 한다면 타성에 젖어있는 선배들은 이런 방식을 거부할지도 모른다. 그렇다면 다음과 같이 유도어트리뷰트를 사용하면 어떨까?



[학과]

학과번호	학과명
1	컴퓨터공학과
2	전자공학과
3	정보통신공학과
4	건축공학과

[학생]

학번	성명	핸드폰	이메일	입학년도	학과번호	일련번호	학번(유도속성)
1	이재학	010-1234-5678	lee@email.com	2011	1	203	
2	김영삼	010-2121-9999	kim@email.com	2009	3	109	
3	노무현	010-3242-1234	no@email.com	2011	2	204	
4	김대중	010-1000-2000	kim2@email.com	2007	2	111	
5	전두환	010-0505-5555	jdh@email.com	2008	4	9	
6	이승만	010-0101-1111	seng@email.com	2010	4	5	

<그림 1.6.4>

실제로 SQL Server에서 물리적으로 구현한다면 다음과 같이 될 것이다.

```
CREATETABLE[dbo].[학생](
        [학번_설계어트리뷰트][int]IDENTITY(1,1)NOTNULL,
        [성명][nvarchar](20)NOTNULL,
        [핸드폰][char](13)NOTNULL,
        [이메일][varchar](50)NOTNULL,
        [입학년도][smallint]NOTNULL,
```

```
        [학과번호][smallint]NOTNULL,
        [일련번호][smallint]NOTNULL,
        [학번_유도어트리뷰트]AS(
                RIGHT('00'+RIGHT(입학년도,2),2)+
                RIGHT('000'+convert(varchar(3),학과번호),3)+
                RIGHT('0000'+convert(varchar(4),일련번호),4)
        )PERSISTED,
        CONSTRAINT[PK_학생]PRIMARYKEYCLUSTERED([학번_설계어트리뷰트]ASC)
)
GO

CREATEUNIQUENONCLUSTEREDINDEX[nix_학번]
ON[dbo].[학생]([학번_설계어트리뷰트]ASC)
GO

INSERT[dbo].[학생](성명,핸드폰,이메일,입학년도,학과번호,일련번호)
VALUES
        ('이재학','010-1234-5678','lee@email.com',2011,1,203)
,       ('김영삼','010-2121-9999','kim@email.com',2009,3,109)
,       ('노무현','010-3242-1234','no@email.com',2011,2,204)
,       ('김대중','010-1000-2000','kim2@email.com',2007,2,111)
,       ('전두환','010-0505-5555','jdh@email.com',2008,4,9)
,       ('이승만','010-0101-1111','seng@email.com',2010,4,5)
GO


SELECT성명,핸드폰,이메일,입학년도,학과번호,일련번호
FROM학생
WHERE학번_유도어트리뷰트='090030109'
```

9자리의 학번도 수용하고, 무결성도 보장하는 방법이다. 굳이 논리 모델에서 구현까지 설명한 이유는 릴레이션의 4가지 특성을 살려도 문제가 없으며, 물리적인 구현에서 문제를 해결 할 수 있는 방법이 존재함을 보이기 위함이었다. 독자들은 쓸데없는 걱정하지 말고 릴레이션의 특성을 꼭 살려서 모델을 만들 수 있어야 한다.

어트리뷰트-다중값, 단일값

단일값은 어트리뷰트 값이 단 하나의 값만을 가지는 것을 말하고, 다중값은 어트리뷰트 값이 2개 이상인 어트리뷰트를 말한다. <그림 1.3.1>를 보자. 학생 릴레이션은 학번, 성명, 핸드폰, 이메일 어트리뷰트를 내포하고 있다. 학번은 기본키이므로 단일값 어트리뷰트다. 성명은 개명신청을 해도

과거의 이름을 사용하지 않는다는 가정이라면 단일값이다. 성명의 경우 만약 데이터베이스를 사용하는 조직이 학교가 아닌 국가라면 이력관리 차원에서 값을 2개 이상 가질 수 있을 것이나, 이는 다중값, 단일값의 문제가 아닌 '성명이력'이라는 이력관리를 위한 릴레이션이 만들어져야 함이 마땅하다. (이력관리의 상세한 설명은 2장에서 다룰 것이다.)

만약 한 명의 학생이 여러 개의 핸드폰 번호를 가지고 있고, 여러 개의 이메일 주소를 가지고 있는 경우에는 핸드폰, 이메일 어트리뷰트가 다중값 어트리뷰트가 된다. 그림으로 도식하자면 다음과 같이 릴레이션이 될 것이다.

학번	성명	핸드폰	이메일
110010203	이재학	010-1234-5678 010-4567-0909	lee@email.com ljk@email.com
090030109	김영삼	010-2121-9999	kim@email.com
110020204	노무현	010-3242-1234	no@email.com

<그림 1.6.5>

다중값의 경우 특별한 경우가 아니고서는 지양해야 한다. 핸드폰이나 이메일을 이용해서 학생을 찾는다거나 하는 업무요건이 있다면 다중값 어트리뷰트는 다음과 같이 다른 릴레이션으로 관리되어야 한다.

학번	성명
110010203	이재학
090030109	김영삼
110020204	노무현

학번	핸드폰
110010203	010-1234-5678
110010203	010-4567-0909
090030109	010-2121-9999
110020204	010-3242-1234

** 기본키= 학번+핸드폰*

학번	이메일
110010203	lee@email.com
110010203	ljk@email.com
090030109	kim@email.com
110020204	no@email.com

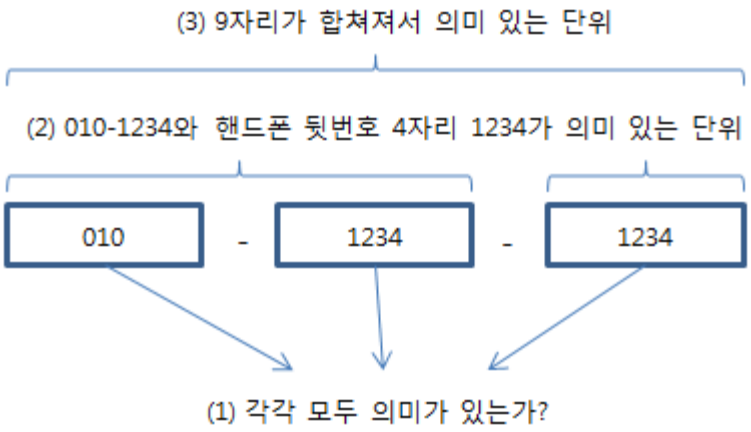
** 기본키= 학번+이메일*

<그림 1.6.6>

이러한 분해 방법은 릴레이션의 특성 중에 하나인 '어트리뷰트의 원자값'과 '1차 정규화'의 훌륭한 예제다. 이 2가지는 각각 다른 절에서 상세히 설명될 것이다.

어트리뷰트-복합어트리뷰트

앞서 릴레이션의 특성에서 '어트리뷰트의 원자값'에 대해서 살펴보았다. '원자값'은 의미 있는 최소 단위를 말한다. 여기서 '의미 있는'에 주목하자. 아래의 <그림1.6.7>에서 (1), (2), (3) 중에 어떤 것이 의미 있는 단위인가?



<그림 1.6.7>

(1)의 경우에 핸드폰 첫 3자리는 저장되는 데이터의 품질을 높이기 위해 핸드폰의 도메인 체크를 위해서 각각을 사용하기도 하였으나 사실 별다른 의미가 없으므로 전화번호 자체를 관리하는 통신사가 아니라면 9자리가 모두 합쳐진 (3)이 의미가 있을 것이다. 하지만 요즘은 미용실 같은 곳에서 핸드폰 뒤 4자리로 고객을 조회하는 경우가 많다. 만약 핸드폰 뒷자리로 검색을 하고, 만약 2명 이상이 검색된다면 이름을 물어보거나 생년월일을 물어봐 식별한다. 이런 경우에는 (2)과 같은 경우가 의미 있는 최소 단위인 원자값이 된다.

이렇게 상황에 따라서 의미 있는 최소단위가 되거나 되지 않을 수 있다. 학번, 전화번호, 생년월일, 날짜(년, 월, 일, 시, 분, 초, 주민등록번호와 같은 어트리뷰트들이 대표적인 복합어트리뷰트가 될 수 있다. 그러므로 복합어트리뷰트는 논리적이라기 보다는 물리적인 것에 가깝다. 사용패턴에 따라서 의미 있는 최소단위가 되기도 하고 더 쪼개져야 하는 상황이 발생할 수도 있다.

외부키(foreign key, 외래키)

외부키는 외부에서 온 다른 릴레이션의 기본키 또는 대체키다. 외부키는 관계(relationship)의 논리적인 표현이다. 표 형태로 데이터를 관리하고자 하는 경우에 관계를 표현하는 방법이 2가지 있는데, 하나는 외부키로 관리하는 방법이고, 다른 하나는 외부키가 아닌 관계를 표현한 릴레이션으로 관리하는 방법이다. <그림 1.6.8>가 관계를 릴레이션으로 표현한 방법이며, <그림 1.6.9>가 외부키로 관계를 표현한 방법이다.

[학생]

학번	성명	핸드폰	이메일
1	이재학	010-1234-5678	lee@email.com
2	김영삼	010-2121-9999	kim@email.com
3	노무현	010-3242-1234	no@email.com

[학과]

학과번호	학과명
1	컴퓨터공학과
2	전자공학과
3	정보통신공학과

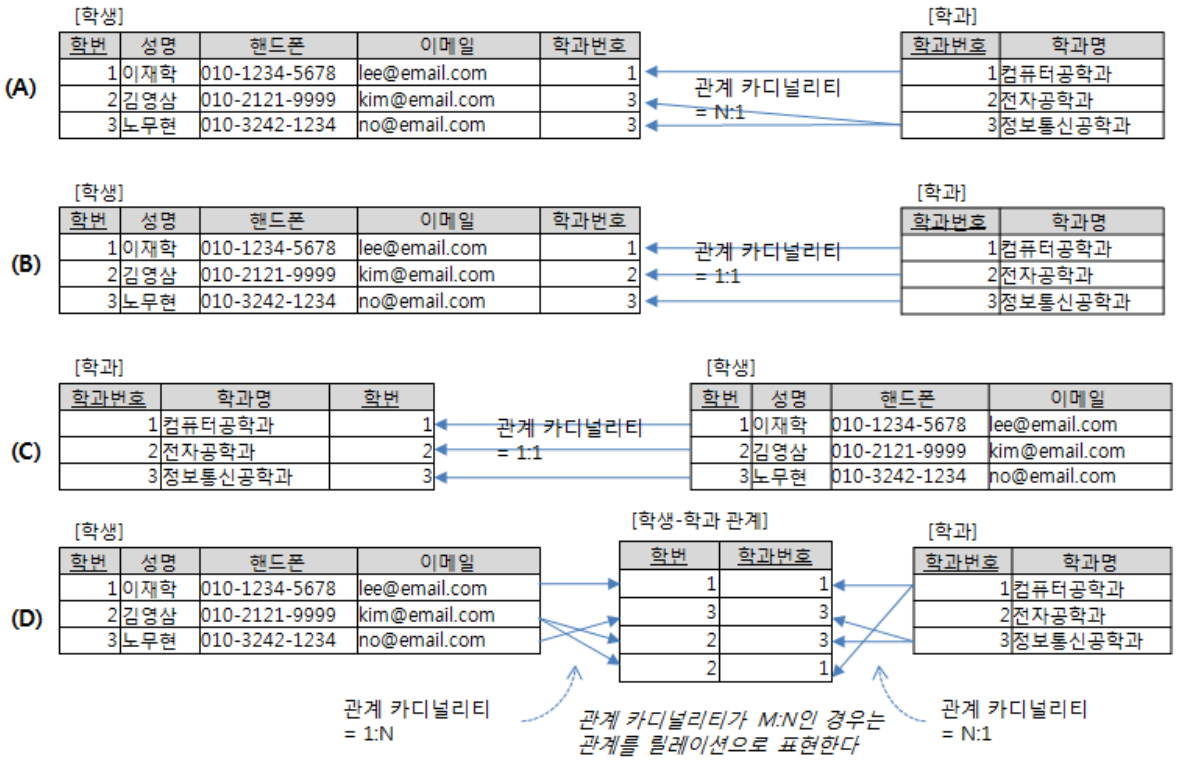
[학생-학과 관계]

학과번호	학번
1	1
3	2
2	3

→ [1,이재학,010-1234-5678, lee@email.com]와 [1, 컴퓨터공학과]이 관계가 있음을 릴레이션으로 표현하였다.

<그림 1.6.8>

릴레이션으로 관계를 관리하는 방법은 고전적인 방식으로 유연성이 매우 뛰어나지만, 관계의 카디널리티(1:1, 1:n)를 고려되지 않아 무결성이 깨질 위험성을 내포하고 있다. 하지만 관계 카디널리티가 m:n의 관계인 경우는 릴레이션형 모델에서는 관계를 릴레이션으로 표현하는 방법 이외에는 다른 표현방법이 없다.

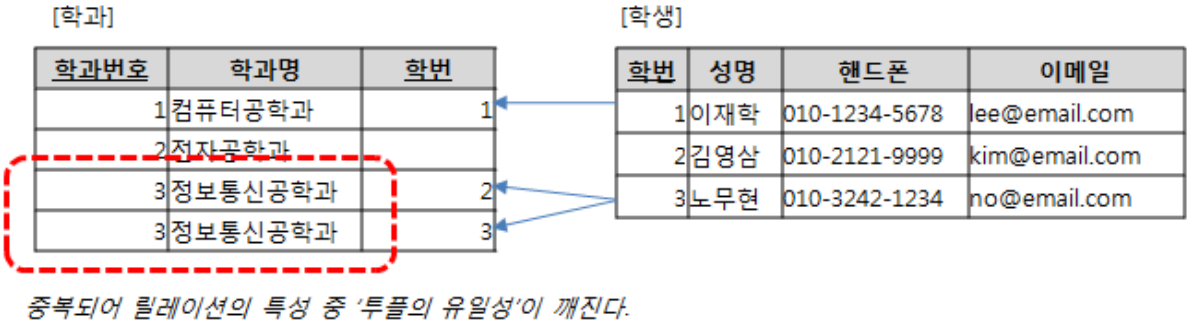


<그림 1.6.9>

릴레이션형 모델의 투플은 포인터가 없기 때문에 어떤 투플들이 어떤 투플과 관계를 맺고 있는지 포인터로는 표현할 방법이 없다. 언뜻 순서로 관계를 표현할 수 있을까 하는 생각도 해 볼 수 있

겠지만, '투플의 무순서'가 릴레이션의 특징이므로 투플의 순서로 학생과 학과가 관계를 맺고 있다는 것을 표현할 수가 없다.

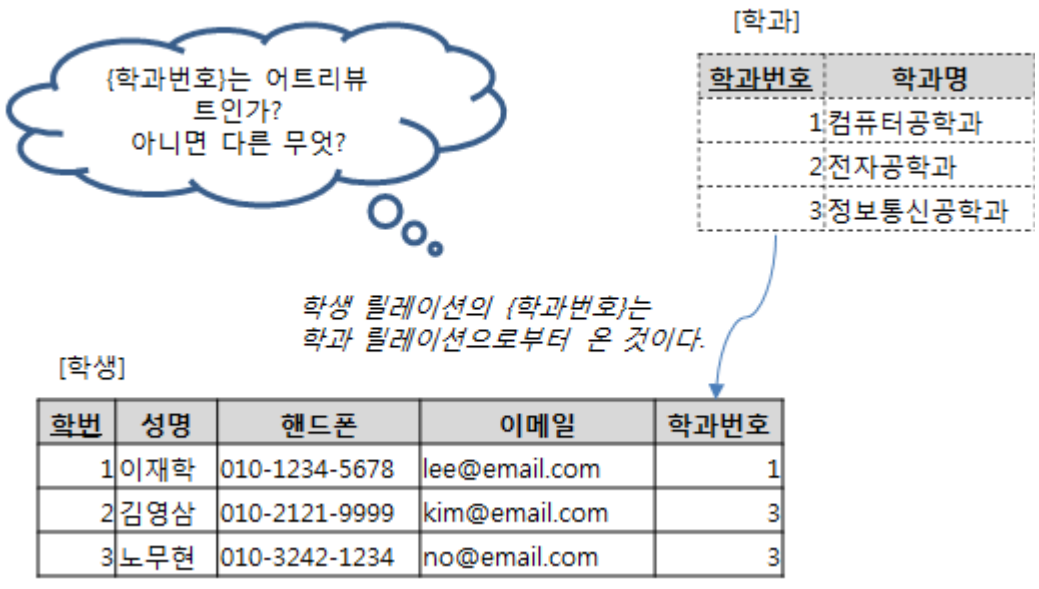
<그림 1.6.9 >의 (A)는 관계를 외부키로 표현한 방법이다. 학생 릴레이션에 외부키가 만들어져서 학생 릴레이션과 학과 릴레이션이 관계를 맺고 있음을 표현하였다. 하지만 왜 학과 릴레이션에 '학번'이 외부키로 붙이지 않았을까? 그 이유는 학생 릴레이션이나 학과 릴레이션 중 한 릴레이션에 학생과 학과가 관계가 있다는 것을 표현해야 하는데, 학과 릴레이션에 어떤 학생들이 소속되어 있는지를 표현하려면 <그림 1.6.10>과 같이 투플들이 반복되게 되어 각각의 투플을 식별하지 못하게 되기 때문이다.



<그림 1.6.10>

그러므로 <그림 1.6.9>와 같이 '투플의 유일성'을 지키면서 관계를 표현하고자 한다면, 학생 릴레이션에 학과 릴레이션의 기본키인 학과번호를 붙여서 각각의 투플들이 어떤 관계를 맺고 있는지 표현할 수 밖에 없는 구조가 되어야 한다.

외부키는 한 가지 논란거리가 있다. 논란거리는 바로 외부키를 어트리뷰트로 볼 것인지 아니면 다른 것으로 인식할 것인지에 대한 문제다. <그림 1.3.7>을 보자.



<그림 1.6.11>

학생 릴레이션에 '학과번호'가 있는데, 이것은 외부키이다. 일부에서는 이를 '관계속성'이라고 명명하여 어트리뷰트의 한 범주로 보고 있다. 하지만 형이상학적 실재론의 관점에서 보면 외부키는 관계의 논리적인 표현이다. 물론 다른 보편자인속성, 종도 각각 별개다. 그러므로 당연히 외부키는 어트리뷰트가 아니며, 어트리뷰트와는 별개로 외부키 그 자체를 특별히 다루어야 한다. 학생은 학과의 존재와 무관하게 이미 존재하며 그 자체다. 속성과 관계를 동시에 지칭하는 보편자는 없다.

널(NULL)

1970년 IBM의 E.F. Code박사는 릴레이션형 데이터 모델을 개발했다. 1985년에는 이상적인 릴레이션형 데이터 모델을 정의하는 12규칙을 발표하였다. 그 중에 3번째 규칙이 null에 대한 규칙이며, 릴레이션형 모델을 지원하는 DBMS는 체계적인 null값 지원을 해야 한다.

- DBMS 는 널 값(알려지지 않거나 적용이 불가능한 데이터)을 사용하기 위해 체계적인 지원.
- 널 값은 디폴트 값과 다르고 어떤 도메인(domain)에 대해서도 독립적.

이처럼릴레이션형 데이터 모델에서는 NULL에 대한 정의를 명확히 했다. 왜 E.F. Codd 박사는 12가지 규칙 (몇 가지 안 되는 규칙 중에서)중에 NULL에 한 자리를 떡 하니 내줬을까? 릴레이션형 데이터 모델은 Legacy 시스템에는 없던 개념이었기 때문이다. 'insert tableA(num) values(1)'를 실행하면 num에는 1 값이 입력된 것이다. 'insert tableA(num) values(null)'의 실행 결과는 num에는 null 값이 입력된 것이다. 만약 num 컬럼에 not null이라고 선언이 되어 있다면 이것이 '무결성'을 위배한 것이 되겠다.

not null 선언된 컬럼에 대한 조회에서 null이 튀어나오는 것이 오류가 아닐까? 숫자1도 독립적인 도메인이 있다. null도 독립적인 도메인이 있다. 왜 null도 독립적인 도메인을 가져야 할까? 생각해보겠다. 다음 글은 참인가?

1과 null은 같을 수도 있고, 다를 수도 있다.

필자가 보기에는 참이다. 비슷한 예로 필자가 A라는 사이트에 회원가입을 하려고 한다. 필자의 이메일 주소는 'admin@databaser.net' 인데, A사이트에는 이메일 항목이 옵션이라 입력을 안 했을 뿐이다. 필자의 관점에서는 이메일 주소가 null이 아니라 'admin@databaser.net' 으로 분명히 값이 있다. 하지만 A사이트의 관점에서 보면 null이다.

그렇다면 null은 무슨 뜻일까? 필자가 보기에 가장 알맞은 뜻은 ' unknown'이다. 왜냐하면 값은 있으나 뭔지 알 수 없을 수도 있고, 아예 값 자체가 없을 수도 있기 때문이다. 그러므로 null은 '' 도 아니고 0도 아니고 ' '도 아니다. 물론 무(無)도 아니다. 그러므로 다른 1 + null의 연산 결과는 null이다. 왜냐하면 알 수 없는 값에 1일 더해봐야 값을 알 수 없기는 마찬가지기 때문이다. 그러므로 null은 다른 어떤 도메인에 대해서도 독립적인 영역에 있어야 한다.

그렇다면 어트리뷰트의 도메인을 결정할 때 null을 허용하는지 안 하는지 어떤 기준에 의해 결정되어야 할까? 역시 정답은 없다. 가장 중요한 것은 현실을 얼마나 잘 반영할 수 있느냐가 아닐까 한다. 회원가입시 A4용지 4~5장 분량의 개인정보를 요청한다면 회원가입을 받는 궁극적인 목적을 달성할 수 있을까? 아마도 아닐 확률이 높을 것이다. 적절한 추상화레벨이 선택되고 우리는 이것을 컴퓨터세계에 반영할 것이다. 하지만 100% 완벽한 추상화 레벨도 없을뿐더러 데이터 통합의 이슈도 있기 때문에 알 수 없는 값이 발생할 수 밖에 없다. 그러므로 반드시 null이 존재해야만 한다.

null은 특별히 관리되어야 한다. 만약 명확한 것을 좋아한다면 null대신 기본값을 세팅하거나 'unknown'인지 아니면 'nothing' 또는 '0' 또는 '', ' '인지 확실히 구별하는 것이 좋다. '알 수 없음'과 같이 애매모호 한 것 보다는 명확하게 구별하는 것이 훨씬 좋다. (DBMS에서 null에 대한 처리 방법은 3장에서 다룰 것임)

참고: Codd의 12규칙

위키 백과에서 "커드의 12 규칙" 으로 검색하면 된다.

기본키(primary key), 후보키(candidate key), 슈퍼키(super key), 대체키(alternate key)

슈퍼키는 릴레이션의 각각의 투플들을 식별할 수 있는 어트리뷰트나 어트리뷰트들의 조합을 말한다. 투플들을 식별할 수 있는 어트리뷰트나 어트리뷰트들의 조합은 하나의 릴레이션에 여러 개 존재할 수 있는데, 이들 중에서 도메인이 단순하며 정보의 양이 적고, null을 허용하지 않는 것들이 후보키가 된다. 후보키는 기본키가 될 후보로써 대표성을 따져 기본키로 선택되게 된다. 기본키의 선택 기준을 정리해 보면 다음과 같다.

- 유일성
- 최소성
- 널(NULL) 허용 여부
- 대표성

예를 들어, 다음의 학생 릴레이션을 보자.

학번	성명	주민번호	우편번호
110010203	이재학	761118-xxxxxxx	123-123
090030109	김영삼	801021-xxxxxxx	456-789
110020204	이재학	820113-xxxxxxx	190-734

<그림 1.6.12>

만약, 성명 + 우편번호가 유일하다면 다음에 열거한 어트리뷰트들이 슈퍼키가 될 것이다.

- 학번
- 주민번호
- 성명 + 우편번호

이들 중에 (성명 + 우편번호)는 유일성과 Not Null이라는 조건을 만족하지만 최소성을 만족하지 못하는 것으로 판단하여 최종적으로 '학번'과 '주민번호'로 후보키가 된다. 마지막으로 후보키들 중에서 대표성을 따져 기본키를 선정해야 한다. 대표성을 따지기도 쉽다. 조직에서 해당 개체를 어떤 방법으로 주로 식별하는지 따져보면 된다. <그림 1.3.8>에 있는 데이터를 학교에서 사용한다면 '학번'이 기본키가 될 것이다. 물론 '주민번호'는 대체키가 된다. 하지만 데이터를 사용하는 조직이 국가기관이라면 당연히 '주민번호'가 기본키기 될 것이다. '학번'은 국가기관에서는 필요 없을 것이므로 당연히 삭제되는 것이 마땅하다 하겠다.

기본키의 선정이 비교적 쉬웠다. 하지만 이는 하나의 릴레이션만 보았을 때다. 관계를 표현한 외부키는 대부분 대체키 보다는 기본키로 만들어지게 된다. 그러므로 선정된 기본키는 관계된 릴레이션에 영향을 끼치게 된다. 릴레이션이 많아지고 관계가 많아지면 그 만큼 복잡해지기 때문에 기본키의 선정이 만만치는 않다. 이러한 고민은 2장에서 다룰 것이니 1장에서는 이론부분만 이해하고 넘어가도록 하자.

제약조건

릴레이션형 모델에서는 최소한의 데이터 품질을 보증하기 위해 다음과 같은 4가지 제약조건을 지켜야 한다. 특히, 앞의 3개는 기본적인 제약조건으로 데이터 품질에 아주 큰 영향을 끼친다. 아래의 제약조건들이 꼼꼼히 따져지지 않고 꼼꼼히 따지더라도 구현되지 않아 데이터 품질 문제가 발생하는 것이다. 제약조건은 데이터 모델이 현실을 제대로 반영하는지에 대한 최소한의 검증도구다. 현실을 제대로 반영하지 않는 것은 바로 실패로 이어진다.

- 도메인 제약조건(domain constraints)
- 개체 무결성 제약조건(entity integrity constraints)
- 참조 무결성 제약조건(referential integrity constraints)
- 업무 제약 조건(business constraints)

도메인 제약조건은 어트리뷰트에 대한 제약조건이며, 엔터티 무결성 제약조건은 릴레이션의 각각의 투플들이 식별 가능함에 대한 제약조건이며, 참조 무결성 제약조건은 관계에 대한 제약조건이다.

도메인 제약조건은 <표 1.2.5>와 같이 데이터 타입, 길이 등의 항목에 대한 제약이다. 만약 '학생' 릴레이션의 '학생번호' 어트리뷰트의 도메인이 <표 1.2.5>와 같이 정의된 경우 다음의 학번이 제약조건을 준수하는지 검토해 보자.

(1) 103530011

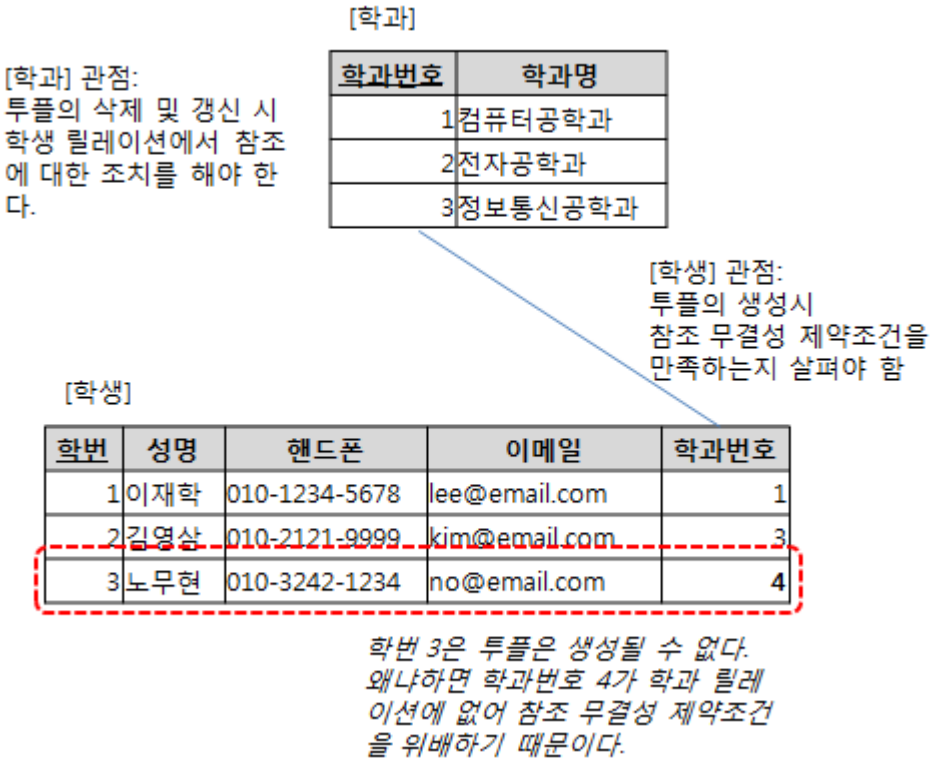
(2) -102720123

(3) 11A301111

(1)의 경우는 3~5자리수인 '353'이 실제로도 학과번호이면 제약조건을 준수하는 것이다. (2)는 음수이므로 제약조건을 벗어난다. (3)의 경우 3번째 자리에 숫자형 문자가 아닌 알파벳이 들어왔으므로 제약조건을 벗어난다.

개체 무결성 제약조건은 릴레이션의 투플들이 유일해야 함을 말하는 것이다. 앞서 슈퍼키, 후보키, 기본키, 대체키에 대해서 알아보았다. 만약 슈퍼키가 존재하지 않는다면 엔터티 무결성 제약조건을 만족시키는 것이 아니다. 하지만 궁극적으로는 슈퍼키보다는 기본키가 있어야만 한다. 왜냐하면 기본키는 널이 아니고 유일성, 최소성, 대표성까지 갖추었으므로 무결성 확보가 용이하기 때문이다.

참조 무결성 제약조건은 릴레이션과 릴레이션과의 관계에 대한 제약조건이다. 관계는 외부키로 논리 모델에서 된다. <그림 1.6.13>에서 보자.



<그림 1.6.13>

여기서 학번 3인 학생 투플이 생성될 수 없는 이유는 2가지다.

(1) 학과번호 4가 학과 릴레이션에 존재하지 않는다.

(2) 학과번호 4는 현실에 존재하지 않는다.

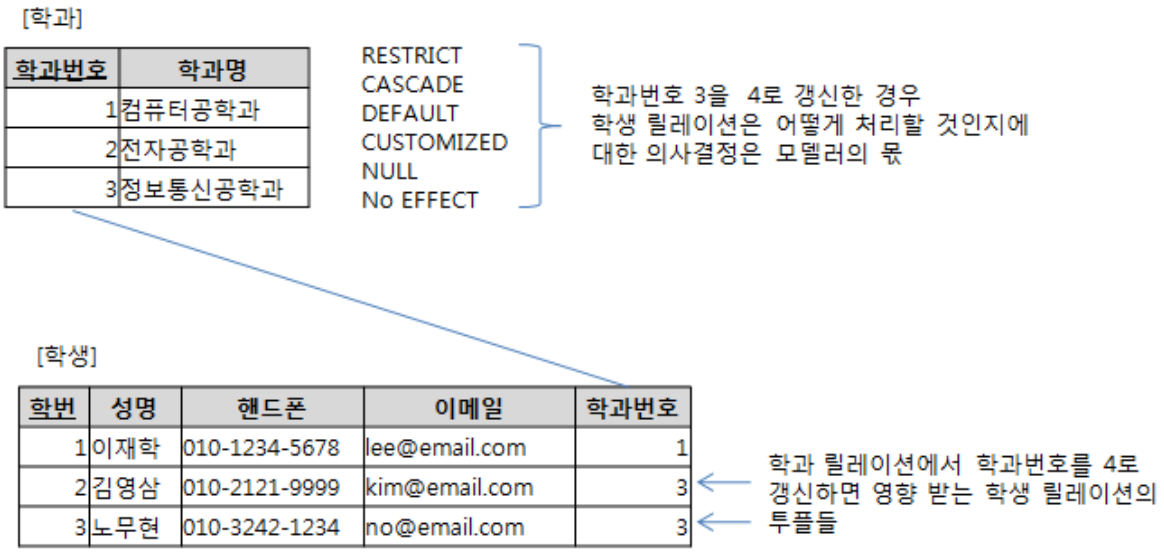
만약 (1)과 같은 문제라면 투플이 생성되는 순서의 문제다. 즉, 학과번호 4가 생성되고 나서 학생 3을 생성하면 된다. 하지만 (2)와 같은 문제라면 현실을 제대로 반영하지 못한 것이다. 입력 참조 무결성의 종류는 다음과 같다.

입력 참조 무결성	학과 릴레이션에 학과번호 4가 존재하지 않으면
DEPENDENT	투플 생성 불가
AUTOMATIC	학과 릴레이션에 학과번호 4를 생성 후 입력
DEFAULT	기본값으로 대체(만약 기본값이 1이면 학과번호 4대신 1일 입력)
CUSTOMIZED	특정한 조건이 만족할 때만 입력 허용
NULL	학생 릴레이션의 학과번호는 NULL값으로 처리
No EFFECT	조건 없이 입력을 허용

<표 1.6.2>

투플의 생성뿐만 아니라 삭제나 갱신의 관점에서 보자. <그림 1.6.13>에서 학생 릴레이션의 삭제는 어떤 제약이 없다. 하지만 학생 릴레이션에서 학과번호를 갱신하려면 반드시 학과 릴레이션에 갱신하려는 학과번호가 존재하는지 검사해봐야 한다. 만약 학과 릴레이션에 갱신하려는 학과번호의 값이 존재하지 않으면 참조 무결성 제약조건 위배다.

학과 릴레이션의 관점에서 삭제와 갱신은 조금 양상이 다르다. 학과 릴레이션에서 학과번호가 3인 투플을 삭제하거나 갱신할 때 <그림 1.6.14>과 같이 학생 릴레이션에서 영향 받는 투플은 2개다.



<그림 1.6.14>

학과 릴레이션을 삭제/갱신 시 참조되는 릴레이션인 학과 릴레이션의 관점에서 참조하는 릴레이션인 학생 릴레이션의 학과번호에 대한 처리를 결정해야 한다. 삭제/갱신 참조 무결성 제약조건은 다음의 <표 1.6.3>와 <표 1.6.4>과 같은 옵션들이 있으며, 업무에 맞게 잘 따져보고 결정하면 된다.

갱신 참조 무결성	학과 릴레이션의 학과번호 3을 4로 갱신 시
RESTRICT	학생 릴레이션에 학과번호 3이 존재하면 갱신 불가
CASCADE	학생 릴레이션의 학과번호=3 인 것들을 모두 4로 갱신
DEFAULT	갱신을 항상 허용하고, 학생 릴레이션의 학과번호=3인 것을 기본값으로 갱신
CUSTOMIZED	특정 조건이 만족할 때만 갱신 허용
NULL	갱신을 항상 허용하고, 학생 릴레이션의 학과번호=3인 것을 NULL값으로 갱신
No EFFECT	조건 없이 갱신을 허용

<표 1.6.3>

삭제 참조 무결성	학과 릴레이션의 학과번호 3을 삭제 시
RESTRICT	학생 릴레이션에 학과번호 3이 존재하면 삭제 불가
CASCADE	학생 릴레이션의 학과번호=3 인 것들을 모두 삭제
DEFAULT	삭제 항상 허용하고, 학생 릴레이션의 학과번호=3인 것을 기본값으로 갱신
CUSTOMIZED	특정 조건이 만족할 때만 삭제 허용
NULL	삭제를 항상 허용하고, 학생 릴레이션의 학과번호=3인 것을 NULL값으로 갱신
No EFFECT	조건 없이 삭제를 허용

<표 1.6.4>

실제로 참조 무결성 구현 시 대부분의DBMS에서 기본값이 RESTRICT라서 인지는모르겠으나 설계자 대부분은RESTRICT로 선택한다. 물론 물리적으로 실제 구현할 때에 이러한 제약조건을 아예 생각지도 않는 경우가 훨씬 더 많다. 이유를 물어보면 공식적으로는 성능이고, 비공식적으로는 불편함으로 귀결된다. 성능상의 이유는 하드웨어의 스펙 문제인데, 돈이 없다는 것도 이유가 되지 않는다. 오히려 참조 무결성 제약조건을 준수하면 프로젝트 전체 비용이 떨어진다. 왜냐하면 현실을 제대로 반영하는 것이기 때문에 디버깅, 테스트 비용이 줄어들고, 제대로 된 데이터 처리 순서를 지켜 동시성 문제가 자연스럽게 해결되어 오히려 성능이 개선 될 수도 있다. 실질적인 이유는 불편함인데, 예를 들어 설명하면 설득된다. 신용불량자에게는 대출을 해 줄 수 없는 제약조건이 있는데, 신용불량자에게 1억 원을 대출해 줘도 시스템에서 막아주지 못하는 것과 같다는 식으로 설득하면 된다.

정의한 제약조건은 성능, 비용, 시간 어떤 것에도 양보할 수 없는 부분이다. 실제 물리적인 구현 시에도 절대 물러섬이 없어야 하며, 만약 어떤 이유에서든 물리적인 구현이 거부되는 일이 발생한다면 반드시 책임에 대한 문서를 받아놔야 한다.

1.7 정규화

정규화는 릴레이션형 모델의 최고의 검증 도구이자 튜닝 도구다. 정규화의 목적을 주로 이상(anomaly) 현상을 없애는 것으로 많이 설명되고 있지만, 자연스럽게 데이터의 중복을 제거한다는 관점에서는 최고의 튜닝 도구다. 또한 종속 관계를 파악함으로써 자연스럽게 데이터 처리 흐름이 정해져서 동시성 문제도 자연스럽게 해결된다. 이번 절에서는 함수적 종속을 먼저 살펴본 후 이상 현상과 1차 ~ 4차 정규형까지 살펴보도록 하겠다. 5차 정규형은 찾아보기도 힘들고 실무에 사용되지도 않으므로 설명하지 않겠다.

함수적 종속과 결정자

난감한 것은 '함수적 종속'이라는 용어가 설명하기도 어렵고, 설명해 봐야 와 닿지 않는다는 것이다. 함수는 흔히 프로그래밍이나 수학에서 많이 사용되는데, 문제는 함수의 내용이 뭔지 설명하는 책이 없다는 것이다. 함수의 내용은 '존재'로 사실 별것 아니다. 예를 들어, A와 B가 있는데, 이 둘은 함수적 종속 관계에 있다고 가정 해보자. 이는 다음과 같이 해석할 수 있다.

> *A가 없으면 B는 존재할 수 없다.*

이 명제가 참이면 B는 A에 함수적 종속된다. 즉, A가 없이 B 단독으로는 의미가 없다는 말이 된다. 다른 예로 아래의 간단한 수식을 살펴보자.

> y = f(x), x, y는 0보다 큰 자연수
> $f(x) = x^2$
> y = 4 라면, x = 2 or -2

y = 4 인 경우 x값은 2 또는 -2다. x에 두 개의 값은 한 번에 대입될 수 없다. 그러므로 y는 x를 결정하지 못한다. 그러나 반대로 x = 2라면 y = 4 가 된다. 즉, x값이 y값을 결정했다. 만약 x= -3 이라면 y값은 9가된다. 역시 x값이 y값을 결정했다. 즉, y는 x에 함수적으로 종속되었다. 이런 함수적 종속은

> x -> y

과 같이 표현하고, "y는 x에 함수적으로 종속된다. 결정자는 x이고, 종속자는 y이다." 라고 이야기한다. y값이 4가 된 것은 x=2 or x = -2와 함수 f()의 내용이 결정을 했다는 것만 알면 정규화는 95% 완료다.

현실에서 함수적 종속을 살펴보자. 어느 고객센터에 전화를 걸고 문의를 할 때 상담원이 고객의 이름을 물으면서 맞냐고 묻는다. 하지만 상담을 진행한다면 필히 휴대전화의 주인이 맞는지 확인을 위하여 주민번호 13자리를 눌러 달라고 하는 경우가 있었을 것이다. 이는 이름만으로는 존재 가치가 없음을 뜻하는 것이다. 이것은 개체 식별의 문제다. 하지만, 여기서 주민번호 13자리와 이름이 조회된 데이터와 다르면 상담원은 주민번호와 이름 중 어떤 것을 신뢰할까? 당연히 상담원

은 주민번호 13자리를 신뢰할 것이며, 본인이 아닌 것으로 판단할 것이다. 왜냐하면 주민번호 13자리가 이름을 결정하기 때문이다. 즉, 함수적 종속을 표현하면 다음과 같을 것이다.

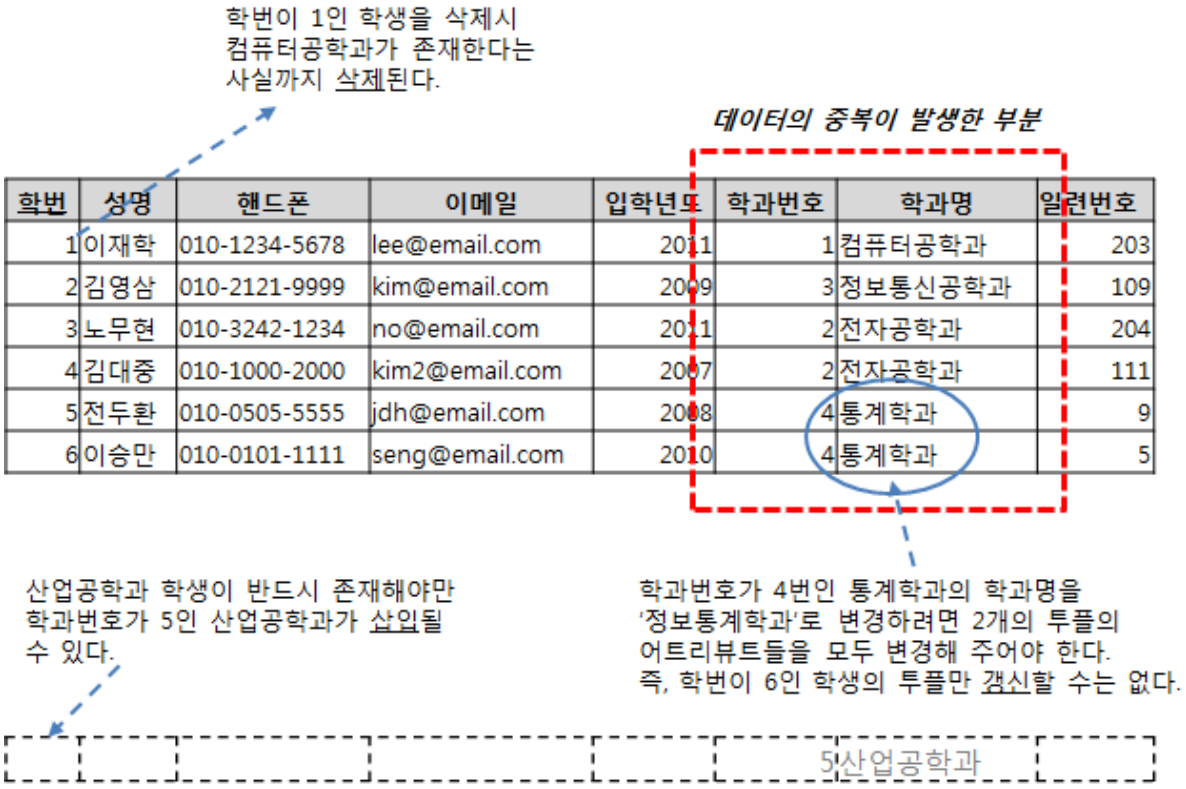
주민번호 -> 이름

주민번호는 결정자이며, 이름은 종속자로 주민번호에 함수적으로 종속된다. 즉, 이름 단독으로는 존재 가치가 없다.

이상(anomaly) 현상

이상 현상은 데이터 중복으로 인해 데이터베이스에 나타나는 모순(현상)이다. 이러한 현상은 데이터의 중복이라는 원인을 없앰으로써 해결할 수 있다.

이상 현상의 종류는 삽입, 삭제, 갱신 이상이 있다. 삽입 이상은 릴레이션에 투플을 삽입 시 각각의 투플의 식별을 할 수 없게 되는 현상을 말한다. 삭제 이상은 릴레이션에서 투플을 삭제 시 원하지 않는 정보까지 삭제되는 현상을 말하며, 갱신 이상은 의도하지 않게 다른 투플의 어트리뷰트들까지 갱신되는 현상을 말한다. 이러한 이상현상은 데이터가 중복된 부분에서 발생한다.



<그림 1.7.1>

갱신이야 2개의 투플이 모두 갱신하면 되지만, 삭제 이상은 정보 손실을 가져온다. 또한 삽입 이상은 정보를 릴레이션에 삽입 할 수 없으므로 역시 정보 손실을 가져온다 할 수 있다. 그래서 실무에서는 가상의 학생을 만들고 Not Null이던 어트리뷰트의 도메인을 NULL을 허용하는 것으로

바꾸고 다음과 같이 삽입과 삭제를 해결하는 경우도 볼 수 있다.

학번	성명	핸드폰	이메일	입학년도	학과번호	학과명	일련번호
-1	가상1	null	null	9999	1	컴퓨터공학과	203
2	김영삼	010-2121-9999	kim@email.com	2009	3	정보통신공학과	109
3	노무현	010-3242-1234	no@email.com	2011	2	전자공학과	204
4	김대중	010-1000-2000	kim2@email.com	2007	2	전자공학과	111
5	전두환	010-0505-5555	jdh@email.com	2008	4	정보통계학과	9
6	이승만	010-0101-1111	seng@email.com	2010	4	정보통계학과	5
-2	가상2	null	null	9999	5	산업공학과	-1

<그림 1.7.2>

이렇게 현실의 변화에 대해서 적응시키다 보면 SQL은 점점 복잡해지고 예외처리, 하드코딩이 들어가게 된다. 근본적인 문제는 데이터의 중복으로 인해 것이다. 물론 기존의 어플리케이션들이 강한 결합(strong coupling)으로 하나를 변경하면 여기 저기서 문제가 발생(부수효과, side-effect)고, 10년 동안 유지보수를 담당하던 담당자가 퇴사를 하여 정규화 하고 싶어도 정규화 할 수 없는 상태일 수도 있다. 그러므로 처음부터 잘해야 한다.

실무에서 이상 현상을 찾지는 않는다. 만약 3정규화 과정을 거친다면 기본적인 이상 현상들이 없어지므로 조금 있다가 살펴볼 보이스-코드(boyce-codd) 정규화와 4 정규화에서의 이상 현상들만 파악할 수 있으면 이상 현상은 졸업이라고 볼 수 있다. 보이스-코드 정규화까지는 함수적 종속을 이용하여 릴레이션을 분해하지만, 4차 정규화는 다중값(multi-value) 종속을 이용하고, 5차 정규화는 조인 종속을 이용한다. 이 책에서는 다중값 종속만 설명하도록 하겠다. 5차 정규화는 사례를 찾기도 어려울뿐더러 필자가 설명할 자신도 없다. 또한 이론서에서 나오는 방식이 아닌 실무적인 방법으로 접근할 것이다. 1차 정규화부터 살펴보도록 하자.

1차 정규화

아래의 <그림 1.7.3>의 A, B타입 릴레이션은 비정규화(non-normalized)된 릴레이션으로 보자. 여기서 비정규화된 릴레이션이란, 1차 정규화되지 않은 릴레이션을 말한다.

A타입

주문자	등급	주문번호	주문일자	분류	상품명	수량	가격
이재학	골드	444	2011-12-14	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500
김대중	실버	333	2011-11-21	철학	칸트의 인간관과 인식존재론 토마스 아퀴나스의 철학	1 1	18,000 11,250
이재학	골드	222	2011-11-16	컴퓨터	데이터베이스 관리론 (양장본 HardCover)	1	21,000
이순신	구리	111	2011-11-04	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500

B타입

주문자	등급	주문번호	주문일자	분류	상품명	수량	가격
이재학	골드	444	2011-12-14	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500
김대중	실버	333	2011-11-21	철학	칸트의 인간관과 인식존재론 토마스 아퀴나스의 철학	2	29,250
이재학	골드	222	2011-11-16	컴퓨터	데이터베이스 관리론 (양장본 HardCover)	1	21,000
이순신	구리	111	2011-11-04	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500

<그림 1.7.3>

A타입은 상품 어트리뷰트에 2개의 어트리뷰트 값, 수량 어트리뷰트에 2개 어트리뷰트 값, 가격 어트리뷰트도 마찬가지로 2개의 어트리뷰트 값을 가진다. 이렇게 하나의 어트리뷰트에 2개 이상의 어트리뷰트 값을 가지는 것을 '반복 그룹'이라고 한다. 어트리뷰트의 관점에서만 보면 '다중값 어트리뷰트'이라고 하고, 릴레이션의 관점에서는 '반복 그룹' 이라고 한다. B타입은 상품 어트리뷰트에 2개의 어트리뷰트 값을 가지지만, 수량과 가격은 집계(aggregation)된 1개의 어트리뷰트 값을 가진다.

A 타입은 수량과 가격은 순서로 구분할 수 밖에 없는 상태다. 만약 그러므로 복잡한 제약 조건이 걸리게 된다. 만약 순서를 보장할 수 없다면 정보는 손실 된다. B타입의 경우도 반복그룹이 존재하지만 주문한 상품이 많아지는 경우는 어떤 상품을 몇 개나 주문했으며 가격은 얼마인지 알 수 없게 된다. 만약 이러한 정보 손실을 막고자 한다면 다음 <그림 1.7.4>과 같이 1차 정규화하면 된다.

주문자	등급	주문번호	주문일자	분류	상품명	수량	가격
이재학	골드	444	2011-12-14	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500
김대중	실버	333	2011-11-21	철학	칸트의 인간관과 인식존재론	1	29,250
김대중	실버	333	2011-11-21	철학	토마스 아퀴나스의 철학	1	11,250
이재학	골드	222	2011-11-16	컴퓨터	데이터베이스 관리론 (양장본 HardCover)	1	21,000
이순신	구리	111	2011-11-04	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500

<그림 1.7.4>

우리는 앞서 <그림 1.7.3>의 릴레이션들을 '일단' 비정규화된 릴레이션으로 보자고 했다. 하지만 추상화 레벨의 차이에 따라 비정규화된 릴레이션이 아닐 수 있다. 업무적으로 보았을 때에 반복 그룹을 원자값으로 취급한다면 1차 정규화된 테이블 될 수 있다.

반복 그룹을 없애는 방법으로 다음과 같은 방법이 있다. 하지만 이러한 릴레이션은 1회 주문당 2개의 상품만 주문할 수 있다는 가정이 있어야 한다. 만약 1회 주문당 3개 이상의 상품을 주문할

수 있게 하려면 릴레이션을 변경해야 한다.

주문자	...	분류	상품명1	수량1	가격1	상품명2	수량2	가격2
이재학	...	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500			
김대중	...	철학	칸트의 인간관과 인식존재론	1	29,250	토마스 아퀴나스의 철학	1	11,250
이재학	...	컴퓨터	데이터베이스 관리론 (양장본 HardCover)	1	21,000			
이순신	...	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500			

<그림 1.7.5>

흔히, 컬럼1, 컬럼2, 컬럼3와 같은 형태로 되는데, '컬럼n' 형태에서 n은 순서를 의미할 수도 있고, 다중값 중에 하나라는 의미도 될 수 있다. 이런 의미에서 본다면 어트리뷰트는 2가지 의미를 가지게 되어 어트리뷰트는 원자값을 가져야 한다는 릴레이션의 특성에 반하게 된다. 만약 '컬럼n'에서 n이 순서를 의미한다면 어트리뷰트의 무순서라는 특성에도 반하게 된다. 이러한 설계 패턴은 명백히 1차 정규형을 위배하지는 않았지만, 의미상으로 릴레이션의 어트리뷰트가 중복된 것이므로 역시 반복 그룹이 존재한다고 봐야 한다.

2차, 3차 정규화

2차 정규화는 반복 그룹이 없고, 기본키에 종속되게 하는 것을 말한다. 아래의 <그림 1.7.6>의 릴레이션의 기본키는 {주문번호 + 상품명}이다.

[주문] 기본키 = 주문번호 + 상품명

주문자	등급	주문번호	주문일자	분류	상품명	수량	가격
이재학	골드	444	2011-12-14	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500
김대중	실버	333	2011-11-21	철학	칸트의 인간관과 인식존재론	1	29,250
김대중	실버	333	2011-11-21	철학	토마스 아퀴나스의 철학	1	11,250
이재학	골드	222	2011-11-16	컴퓨터	데이터베이스 관리론 (양장본 HardCover)	1	21,000
이순신	구리	111	2011-11-04	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500

<그림 1.7.6>

함수적 종속성이 있는지 알아보는 방법은 삭제했을 때 다른 어트리뷰트들이 존재 가치가 있는지 또는 변경해보았을 때 다른 어트리뷰트들도 함께 변경되어야 현실과의 일치성이 있는지를 살펴보는 것이다. 만약 아래의 <그림 1.7.7>과 같이 주문번호와 상품명이 제거 되었다면 어떤 어트리뷰트들이 영향을 받을까?

[주문] 기본키 = 주문번호 + 상품명

주문자	등급	주문번호	주문일자	분류	상품명	수량	가격
이재학	골드	444	2011-12-14	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500
김대중	실버		2011-11-21	철학		1	29,250
김대중	실버	333	2011-11-21	철학	토마스 아퀴나스의 철학	1	11,250
이재학	골드	222	2011-11-16	컴퓨터	데이터베이스 관리론 (양장본 HardCover)	1	21,000
이순신	구리	111	2011-11-04	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500

<그림 1.7.7>

- 주문자: 어떤 상품을 주문했는지 알 수 없게 된다.
- 등급: 영향 없음.
- 주문일자: 언제 어떤 상품을 주문했는지 알 수 없게 된다.
- 분류: 어떤 상품에 대한 분류인지 알 수 없게 된다.
- 수량: 어떤 상품을 몇 개 주문했는지 알 수 없게 된다.
- 가격: 어떤 상품에 대한 가격인지 알 수 없게 된다.

여기서 등장하는 단어를 보면 '상품'이 압도적으로 많다. 그러므로 영향을 가장 많이 끼치는 어트리뷰트인 상품을 먼저 살펴보는 것이 릴레이션을 단순하게 만드는 방법일 것이다. 그러므로 아래의 <그림 1.7.8>과 같이 'HADOOP 완벽 가이드(개정판)'이 '논리적 독서법'으로 변경되었다고 가정해보자.

주문자	등급	주문번호	주문일자	분류	상품명	수량	가격
이재학	골드	444	2011-12-14	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500
김대중	실버	333	2011-11-21	철학	칸트의 인간관과 인식존재론	1	29,250
김대중	실버	333	2011-11-21	철학	토마스 아퀴나스의 철학	1	11,250
이재학	골드	222	2011-11-16	컴퓨터	데이터베이스 관리론 (양장본, HardCover)	1	21,000
이순신	구리	111	2011-11-04	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500

주문상품이 '논리적 독서법'으로 바뀐다면?

주문자	등급	주문번호	주문일자	분류	상품명	수량	가격
이재학	골드	444	2011-12-14	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500
김대중	실버	333	2011-11-21	철학	칸트의 인간관과 인식존재론	1	29,250
김대중	실버	333	2011-11-21	철학	토마스 아퀴나스의 철학	1	11,250
이재학	골드	222	2011-11-16	컴퓨터	데이터베이스 관리론 (양장본, HardCover)	1	21,000
이순신	구리	111	2011-11-04	컴퓨터	논리적 독서법	1	31,500

<그림 1.7.8>

'HADOOP 완벽 가이드(개정판)'이 '논리적 독서법'으로 변경되었음에도 불구하고 분류는 '컴퓨터'이고 가격이 '31,500'인 것은 어색하다. 왜냐하면 '논리적 독서법'의 분류는 '인문' 또는 '독서' 쯤이 될 것이고, 가격도 '논리적 독서법'의 가격인 '9,000'으로 변경되어야 현실과의 일치성이 있기 때문이다. 그러므로 상품이 변경되었다면 아래의 <그림 1.7.9>에서 볼 수 있듯이 분류는 '컴퓨터'에서 '독서'로, 가격은 '31,500'에서 '9,000'으로 변경되어야 한다.

주문자	등급	주문번호	주문일자	분류	상품명	수량	가격
이재학	골드	444	2011-12-14	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500
김대중	실버	333	2011-11-21	철학	칸트의 인간관과 인식존재론	1	29,250
김대중	실버	333	2011-11-21	철학	토마스 아퀴나스의 철학	1	11,250
이재학	골드	222	2011-11-16	컴퓨터	데이터베이스 관리론 (양장본, HardCover)	1	21,000
이순신	구리	111	2011-11-04	컴퓨터	HADOOP 완벽 가이드(개정판)	1	31,500

주문상품이 '논리적 독서법'으로 바뀐다면?

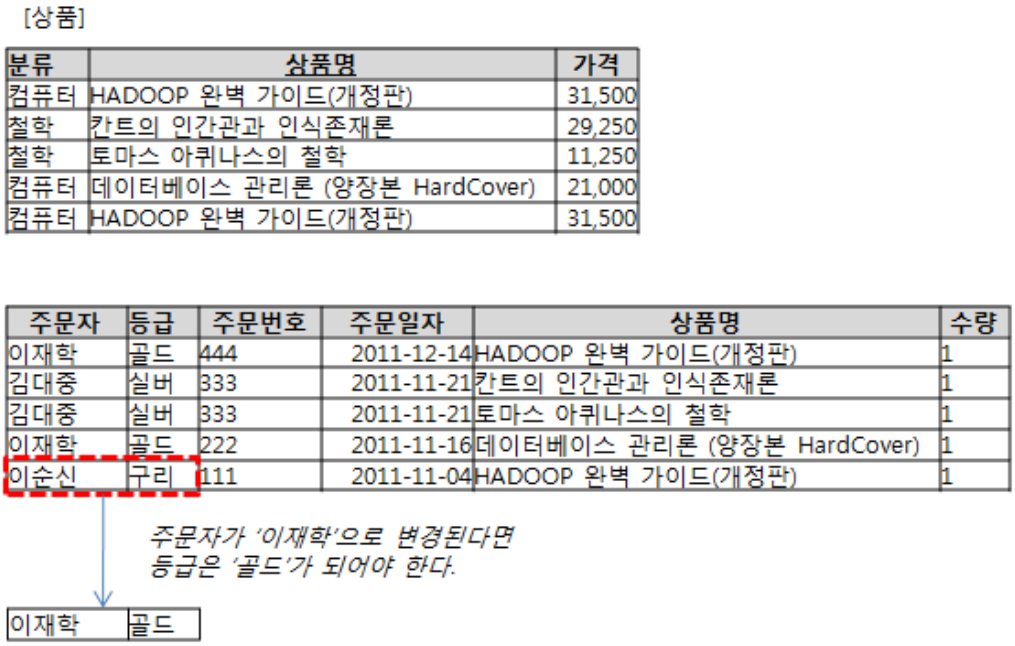
독서	논리적 독서법	9,000

<그림 1.7.9>

상품명이 변경됨에 영향을 받아 변하는 어트리뷰트들이 있다면 그 어트리뷰트들이 종속자이고, 변경된 어트리뷰트는 결정자이다. 이를 다음과 같이 함수적 종속을 표현할 수 있다.

- 상품명 -> 분류
- 상품명 -> 가격

릴레이션은 함수적 종속에 의해 <그림 1.7.10>과 같이 분해 될 수 있다.



<그림 1.7.10>

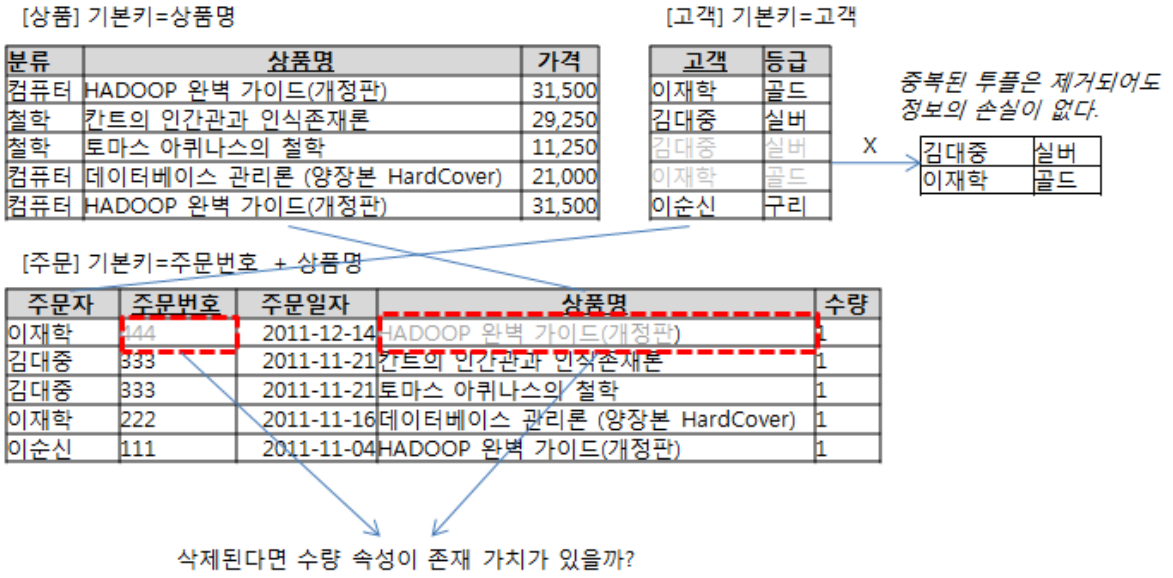
만약 주문번호 111에 주문자의 등급을 알고자 한다면 주문번호 111에 해당하는 고객이 누군지 알고 나서야 등급을 알 수 있다. 이론적으로 말하면 C를 알고자 한다면 먼저 B를 알아야 그 다음에 C를 알 수 있는 것이다. 이러한 종속을 이행 종속(transitive dependency)이라고 하고, 다음과 같이 표현한다.

A -> B -> C

또한 기본키는 {주문번호 + 상품명} 인데도 불구하고, 주문번호만 알 수 있다면 해당 주문에 대한 고객과 고객의 등급을 알 수 있으므로 다음과 같은 종속성이 있음을 알 수 있다.

- 주문자 -> 등급
- 주문번호 -> 주문자 -> 등급

그러므로 <그림 1.7.10>과 같이 주문자 어트리뷰트 값이 '이재학'으로 변경된다면 등급의 어트리뷰트 값은 '골드'가 되어야 현실과 일치한다. 위와 마찬가지 방법으로 상품, 고객, 주문과 같이 분해되었다. 릴레이션 분해 시 중복된 투플은 제거된다.

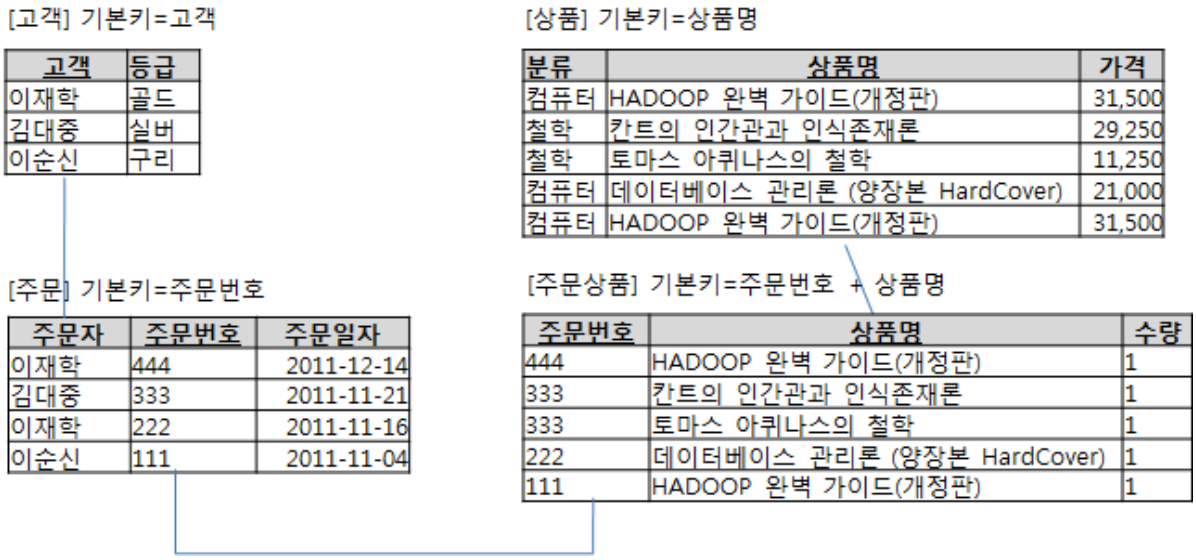


<그림 1.7.11>

주문 릴레이션을 보면 주문번호가 기본키가 되어 중복된 투플이 없어야 함에도 불구하고 주문번호가 중복되는 것을 볼 수 있다. 주문 릴레이션의 후보키는 [주문번호 + 상품명], [주문자 + 주문일자 + 상품명] 이다. 만약 <그림 1.7.11>과 같이 주문번호와 상품명을 삭제한다면 수량 어트리뷰트는 존재의 의미가 없어진다. 또한 주문번호를 알면 주문자와 주문일자도 알 수 있다. 함수적 종속을 정리해 보면 다음과 같다.

- 주문번호 + 상품명 -> 수량
- 주문자 + 주문일자 + 상품명 -> 수량
- 주문번호 -> 주문자
- 주문번호 -> 주문일자

함수적 종속에 의해 다음과 같이 릴레이션이 분해되었다. 만약 고객이 상품을 주문할 당시의 고객등급을 알고 싶다면, 주문 릴레이션에 등급 어트리뷰트를 추가할 필요가 있을 것이다.



<그림 1.7.12>

이제 3차 정규화가 끝났다. 데이터 모델링이나 설계를 조금이라도 해 본 경험이 있다면 이러한 방식이 과연 필요가 있을까 싶기도 할 것이다. 필자의 생각으로는 조금은 필요하다고 본다. 왜냐하면 Top-Down 방식이 아닌 Bottom-Up 방식을 사용할 때는 정규화가 필요하기 때문이다. 하지만 기본적으로 릴레이션, 어트리뷰트, 릴레이션쉽이 파악되면 대부분 3차 정규화와 가까워지므로 어트리뷰트가 많은 릴레이션에 대해서 정규화를 시행하면 될 것이다.

보이스-코드(Boyce-Codd) 정규화

이제 함수적 종속의 마지막 정규형인 BCNF(Boyce-Code Normal Form)에 대해 살펴보도록 하자. 위의 예제는 함수적 종속성이 더 이상 없으므로 BCNF를 설명할 수 없다. 그러므로 보이스-코드 정규화는 다른 예제를 가지고 진행하겠다.

BCNF는 "모든 결정자가 후보키이면 BCNF이다"가 정의의 끝이다. 후보키인지 아닌지를 따져보려면 직접 데이터를 살펴보거나 업무 규칙을 파악하여 유일성만 보장되는 컬럼이나 컬럼의 조합이면 된다. 이것이 BNCF의 전부다. 단, 기본 컬럼이 3개 이상이고, 2개 이상의 어트리뷰트가 후보키(복합키)가 2개 이상인 경우에만 보이스-코드 정규화의 대상이다. 다음의 <그림 1.7.13>을 보자.

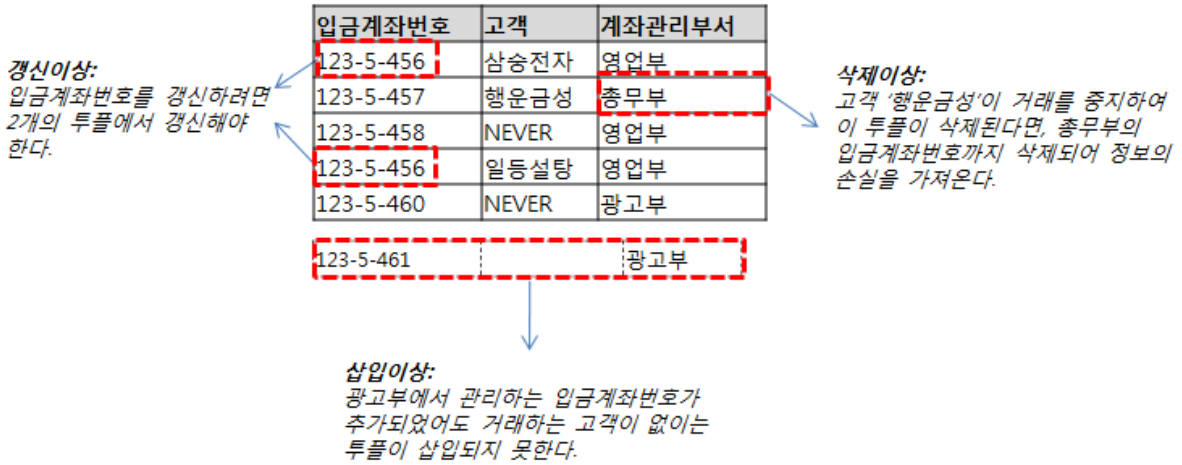
<그림 1.7.13>

<그림 1.7.13>의 릴레이션은 고객과 거래 후 고객에게 관리부서별로 입금계좌번호를 알려주며,

각각의 계좌번호를 관리하는 부서가 다른 경우다. 함수적 종속성을 보자.

- 입금계좌번호 + 고객 -> 관리부서
- 관리부서 -> 입금계좌번호

모든 결정자가 후보키이면 BCNF다. 하지만 {관리부서 -> 입금계좌번호}는 후보키가 아니므로 BCNF가 아니다. 이 릴레이션에서는 다음 <그림 1.7.14>와 같은 이상 현상들이 발생한다.



<그림 1.7.14>

위와 같은 이상 현상들을 없애기 위해서는 <그림 1.7.15>와 같이 릴레이션을 분해하여야 한다.

입금계좌번호	고객
123-5-456	삼숭전자
123-5-457	행운금성
123-5-458	NEVER
123-5-456	일등설탕
123-5-460	NEVER

입금계좌번호	계좌관리부서
123-5-456	영업부
123-5-457	총무부
123-5-458	영업부
123-5-460	광고부

<그림 1.7.15>

4차 정규화

4차 정규형은 보이스-코드 정규형이면서, 다중값종속(multi-value dependent)가 없는 릴레이션을 말한다. 다중값종속이란 하나의 값이 여러 값을 결정하는 것을 말한다. "->>"과 같이 표기한다. 이 릴레이션에는 한 학생이 여러 특별활동을 할 수 있고, 복수전공이 가능하다는 제약조건이 있다. 다중값 어트리뷰트가 2개 이상 존재하는 릴레이션이거나 3원 릴레이션쉽일 경우 4차 정규화의 대상인지 살펴보아야 한다.

[특별활동] 기본키 = 학생번호 + 특별활동 + 전공

학생번호	특별활동	전공
100	스키	경영
100	수영	경영
100	스키	컴공
100	수영	컴공
200	골프	수학
300	수영	음악

<다치 종속>
* 학생번호 ->> 특별활동
* 학생번호 ->> 전공

<그림 1.7.16>

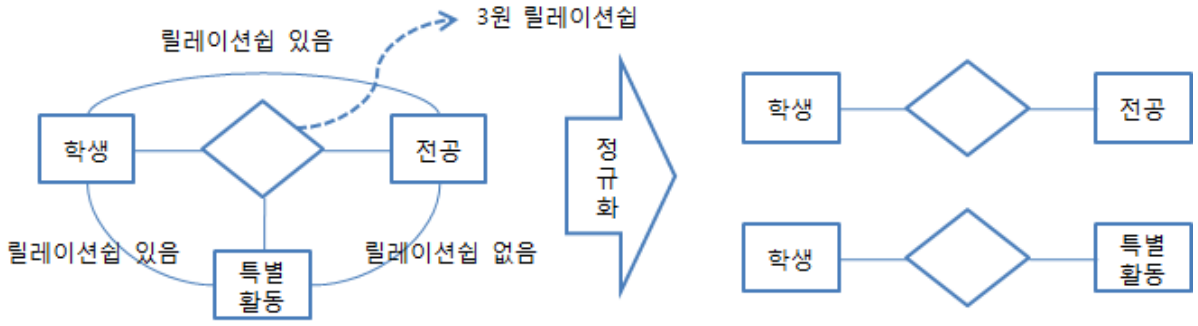
다중값 종속을 제거하려면 릴레이션을 다음과 같이 분해 해야 한다.

학생번호	특별활동
100	스키
100	수영
200	골프
300	수영

학생번호	전공
100	경영
100	컴공
200	수학
300	음악

<그림 1.7.17>

이런 현상이 발생하는 이유는 릴레이션쉽이 제대로 파악되지 않았기 때문이다. 또한 어떤 학생이 어떤 특별활동과 어떤 전공을 하는 지에 대한 2가지의 주제를 하나의 릴레이션에 표현했기 때문에 발생한 현상이다. 그림으로 표현하면 다음과 같다. 3원 릴레이션쉽을 제거하라고 모델링 책에서 떠드는 이유가 다중값종속 때문이기도 하다.



<그림 1.7.18>

흔한 경우는 아니지만, "특별활동"과 "전공"도 관계를 가진다면, 3원 릴레이션쉽을 그대로 둬도 괜찮지만, 5차 정규화의 이슈가 있다. 그러므로 3원 관계는 3개의 릴레이션의 릴레이션쉽을 따져보고 2원 릴레이션으로 만들어주는 것이 속 편하다. 2원 릴레이션으로 만들다 보면 자연스럽게 "특별활동"과 "전공"이 엮일 업무적인 이유를 찾지 못해서 자연스러운 4차 정규형이 될 것이다.

직관적인 중복 제거

사실 몇 정규화가 필요하고, 몇 정규형인지 아는 것은 중요하지 않다. 중요한 것은 중복 제거에 있다. 이상 현상들은 데이터의 중복에 비롯된 것이다. 여러 모델러들과 마찬가지로 필자도 트래이닝이 되어서 그런지 몰라도 일련의 과정을 거치지 않고도 4차 정규화까지 직관적으로 가능하다. 필요하다면 함수적 종속 다이어그램을 그려보면 답이 명확해 진다.

아래의 <그림 1.7.19> 릴레이션은 게임에서 캐릭터가 미션을 완료하고 아이템을 보상받은 것을 표현했다. 캐릭터가 미션을 부여 받고, 미션에 대한 진행상태를 표현한 릴레이션으로 미션 완료 시 주어지는 보상 중 골드는 미션마다 정해져 있고, 다른 아이템은 랜덤으로 n개 주어지는 업무 규칙이다.

캐릭터번호	미션번호	완료상태	보상
120	1	완료	1000골드
120	1	완료	칼+4
120	2	패스	보상없음
500	1	완료	1000골드
500	1	완료	방패+2
500	1	완료	칼+2
500	2	완료	2000골드
500	2	완료	활+5
500	3	패스	보상없음
500	4	패스	보상없음

<기본키>
계정키+부여미션+보상

부여된 미션을 '패스'하면
보상이 없음

<그림 1.7.19>

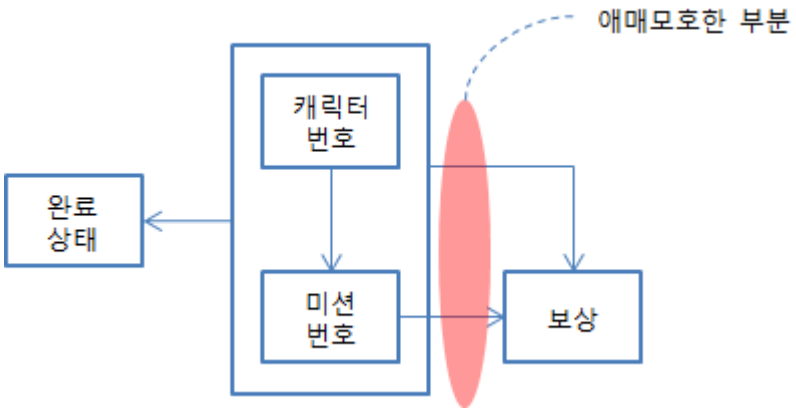
이 릴레이션은 다음과 같은 복잡성이 숨겨져 있다.

- 릴레이션에 계정키가 120인 캐릭터번호 3번이 미션을 완료했다는 정보 삽입 시 보상된 아이템에 대한 투플도 삽입해야 한다.
- 미션 부여시 완료상태가 '진행중'이라는 값으로 세팅되고, 미션 완료 시 아이템을 3개 이상 보상받는다면, 하나의 투플은 갱신하고, 다른 두 개의 투플은 삽입되어야 한다.
- 보상되는 아이템의 하나는 부여된 미션에 의해 결정되지만, 나머지 하나는 랜덤으로 보상되는 아이템이다.

이 릴레이션은 다음과 같은 종속성이 존재한다.

- 캐릭터번호 -> 미션번호
- 미션번호 -> 보상(일부, 예: 미션번호=1 -> 1000골드)
- 캐릭터번호 + 미션번호 -> 예: 보상(일부, 미션번호=1 -> 칼+2, 방패2)
- 캐릭터번호 + 미션번호 -> 완료상태

함수적 종속 다이어그램을 그려보면 다음과 같다.



<그림 1.7.20>

함수적 종속 다이어그램을 보면 기본키의 일부에 종속되는 부분 종속이 존재함을 알 수 있다. 또한, 다중값 어트리뷰트가 2개 이상 존재하는 것 같아 보이기도 한다. 이게 2차 정규화 대상인 4차 정규화 대상인지도 모호하다. 하지만 데이터 중복 관점에서 함수적 종속성을 따져보면 어느 정도 명확해 짐을 알 수 있다. 정규화는 함수적 종속에 따라 다음과 같이 릴레이션을 분해하면 된다.

캐릭터번호	미션번호	완료상태	보상
120	1	완료	1000골드
120	1	완료	칼+4
120	2	패스	보상없음
500	1	완료	1000골드
500	1	완료	방패+2
500	1	완료	칼+2
500	2	완료	2000골드
500	2	완료	활+5
500	3	패스	보상없음
500	4	패스	보상없음

정규화

[미션]

미션번호	보상골드
1	1000골드
2	2000골드
3	3000골드
4	4000골드

[캐릭터-미션]

캐릭터번호	미션번호	완료상태
120	1	완료
120	2	패스
500	1	완료
500	2	완료
500	3	패스
500	4	패스

[보상]

캐릭터번호	미션번호	보상
120	1	칼+4
500	1	방패+2
500	1	칼+2
500	2	활+5

<그림 1.7.21>

또 다른 예로 <그림 1.7.22>를 보자. 이 릴레이션은 각각의 사원의 부양가족과 자격증을 표현했

다. 각각의 사원은 n명의 부양가족이 있을 수 있으며, n개의 자격증을 소유할 수 있다.

사원번호	부양가족	부양가족관계	자격증
1	이재학	부	OCP
1	김영임	모	OCP
1	이재학	부	정보처리
1	김영임	모	정보처리
2	심현규	부	MCDBA
2	뺑덕어멈	모	MCDBA
2	심현규	부	CCNA
2	뺑덕어멈	모	CCNA

<그림 1.7.22>

2개의 다중값 어트리뷰트가 존재하고, 그냥 봐도 부양가족과 자격증과는 아무런 관련이 없음을 알 수 있다. 그러므로 이 릴레이션은 다음과 같이 4차 정규화 될 수 있다.

사원번호	부양가족	부양가족관계
1	이재학	부
1	김영임	모
2	심현규	부
2	뺑덕어멈	모

사원번호	자격증
1	OCP
1	정보처리
2	MCDBA
2	CCNA

<그림 1.7.23>

정규화 정리

1차 ~ 4차 정규화를 정리해 보자.

- 1차 정규화 – 반복 그룹의 제거
- 2차 정규화 – 2개 이상의 어트리뷰트로 구성된 기본키에 부분 종속 제거
- 3차 정규화 – 이행 종속 제거
- BCNF 정규화 – 모든 결정자가 후보키가 되도록 릴레이션 분해
- 4차 정규화 – 3원 릴레이션쉽 또는 2개 이상의 다중값 어트리뷰트의 다중값 종속 제거

이상 현상은 의도치 않게 정보가 손실되거나, 다른 투플들도 변경해 주어야 하는 일종의 부수효과(side-effect)를 말한다. 실무에서는 삽입, 삭제, 갱신 이상을 굳이 찾지 않는다. 다만, 이론을 실무에 적용할 때에 정보 손실이나 부수적인 변경이 뒤따른다면 이상 현상이 있으니 주의 깊게 살펴봐서 정규화 하면 된다.

흔히 실무에서 3차 정규화 과정까지 정규화를 해야 한다고 하기도 하고 보이스-코드 정규화 과정까지 거쳐야 한다고 이야기하기도 한다. 함수적 종속에 의한 중복 제거는 BCNF이면 더 이상은

존재하지 않으므로 BCNF로 만드는 것은 좋은 기준이 될 수 있을 것이다. 4차 정규화로 넘어가면 다중값 종속, 5차 정규화는 조인 종속을 다뤄야 한다. 5차 정규화는 실무에 쓰이지도 않을뿐더러 찾아보기도 어렵다.

필자가 어렵게 정규화를 설명하기는 했지만, 실제로 하향식(top-down)방식에서는 이러한 방법으로 정규화를 수행하지는 않는다. 왜냐하면 릴레이션, 어트리뷰트, 릴레이션쉽을 잘 파악한다면 3정규형까지는 자연스럽게 만들어지기 때문이다. 설명된 방식은 상향식(bottom-up) 방식에서는 써먹어 볼만 하지만, 역시 형이상학적 실재론의 관점으로 세상을 바라본다면 쉽게 정규화된 릴레이션을 도출할 수 있다.

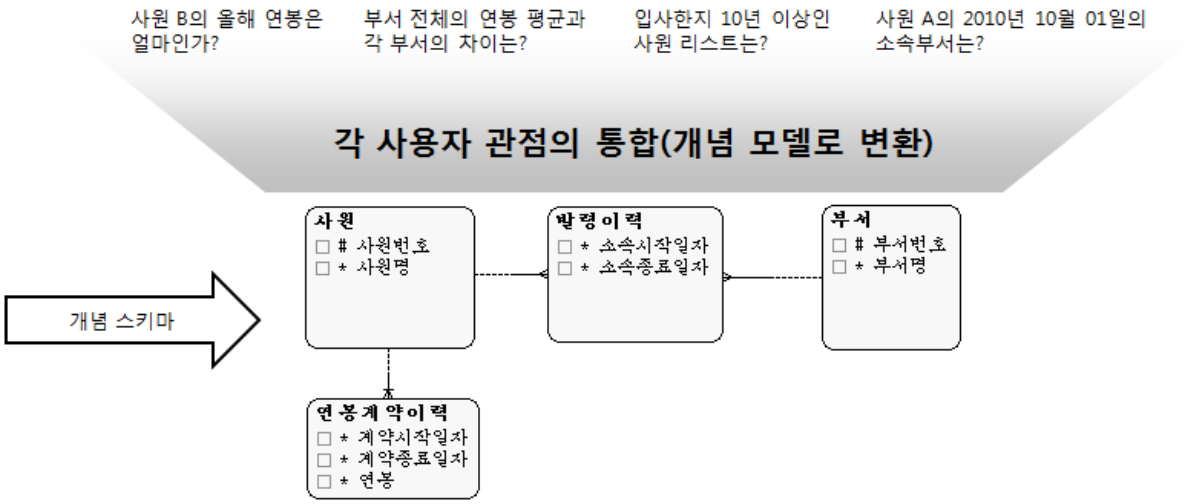
정규화의 궁극적인 목적은 데이터베이스의 '데이터 중복의 최소화'다. 데이터의 중복을 없앰으로써 불필요한 처리비용을 없애고, 여러 가지 이상 현상들을 제거하여 데이터의 불일치나 무결성을 굳건히 지키는 최고의 튜닝도구다. 앞서서 골치 아픈 이론을 조금 설명하였지만 결론적으로 보면 범주화를 잘하고, 중복을 어떤 방법이건 간에 없애면 된다. 몇 차 정규화인지 맞추는 것이 중요한 것이 아니라 데이터의 중복과 이상 현상을 제거하는 것이 중요하다.

1.8 삼단계 스키마 구조

사용자들은 각자 다른 관점에서 각자 다른 업무를 진행한다. 같은 팀이라도 요구하는 데이터는 상이할 수도 있고, 일부만 다를 수도 있고, 같을 수도 있다. 예를 들어, 다음과 같은 요구사항이 있다고 가정하자.

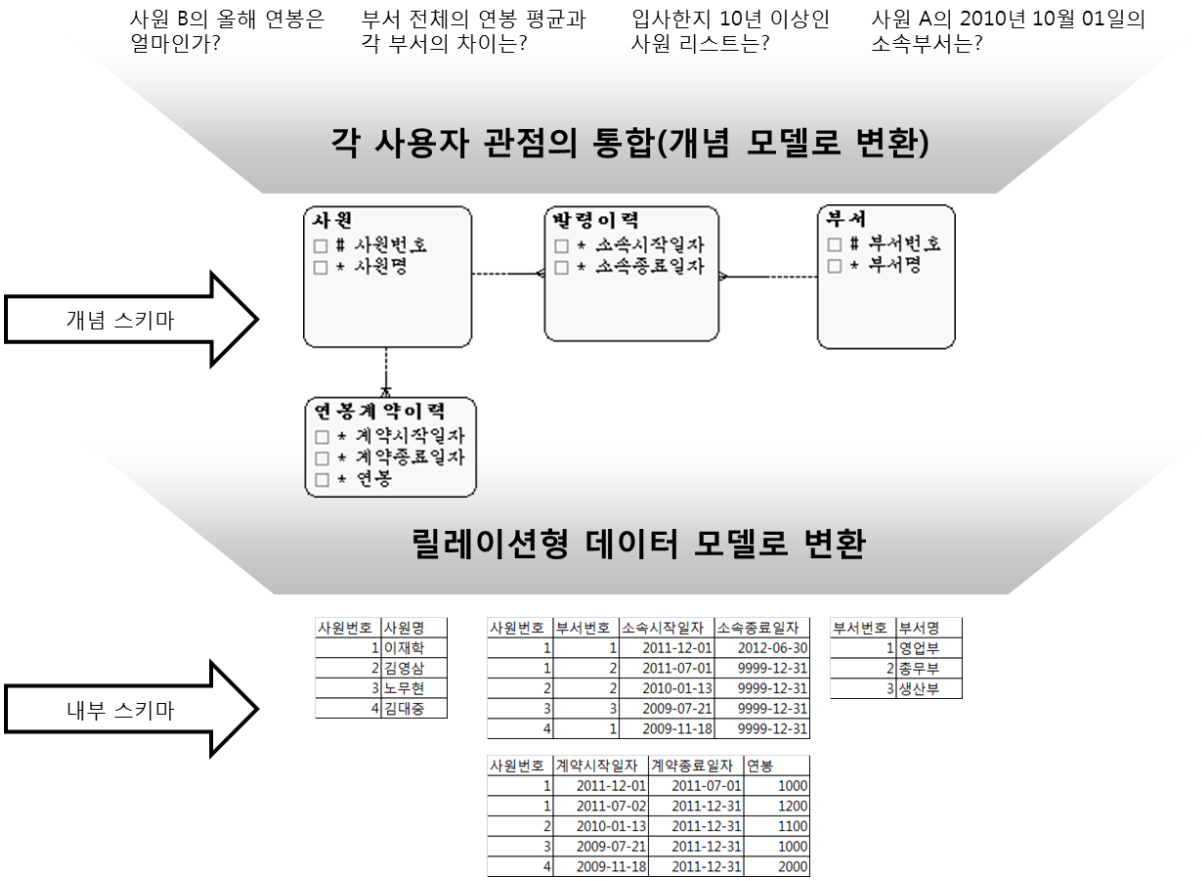
- 사원 B 의 올해 연봉은 얼마인가?
- 부서 전체의 연봉 평균과 각 부서의 차이는?
- 입사한지 10 년 이상인 사원 리스트는?
- 사원 A 의 2010 년 10 월 01 일의 소속부서는?

데이터베이스 구축은 이렇게 다양한 요구사항을 반영해야 하는데, 데이터베이스의 정의에 따라 다수의 사용자들이 공유 할 수 있도록 통합시켜야 한다.



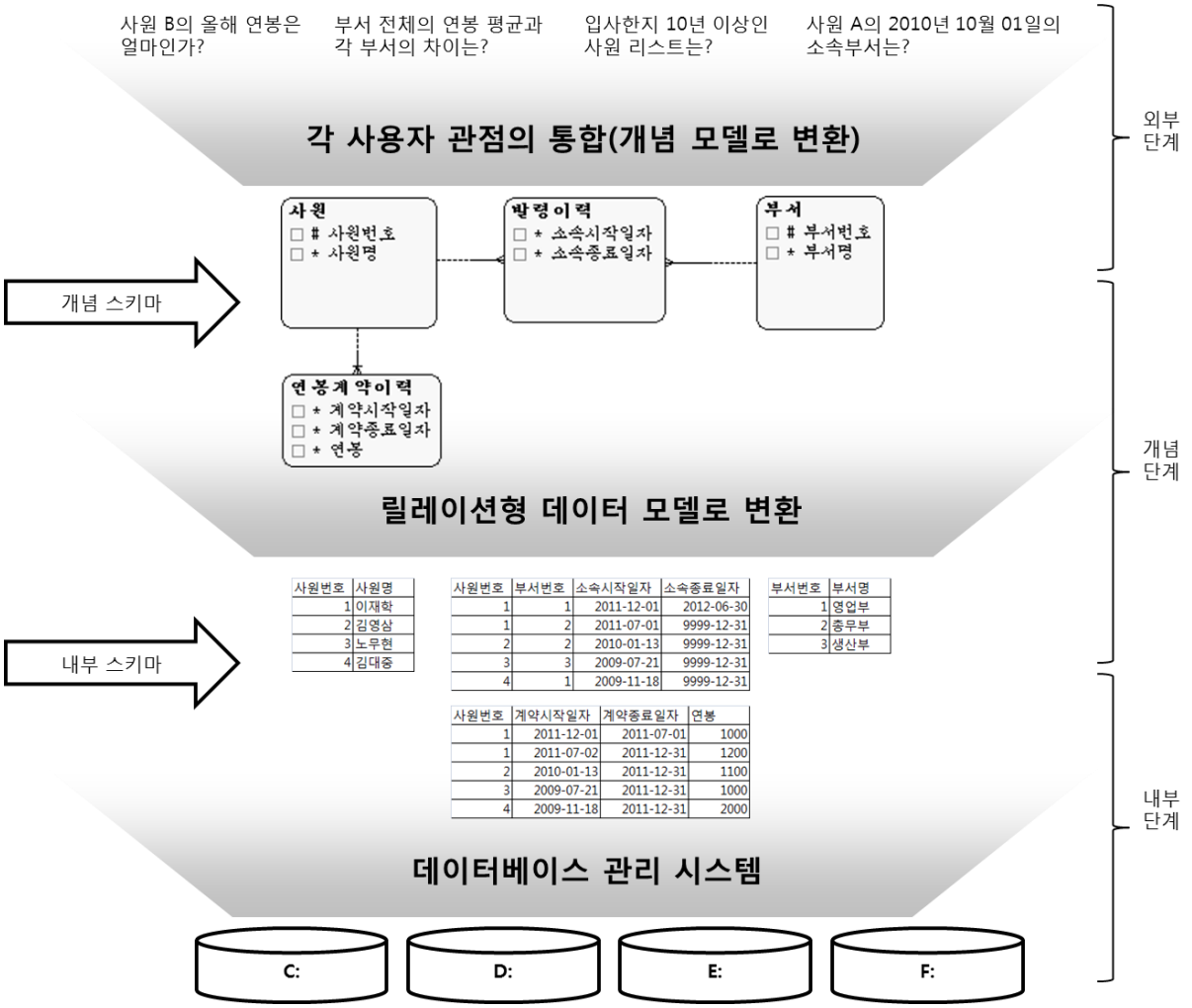
<그림 1.8.1>

이렇게 전체 관점으로 통합된 스키마가 개념 스키마다. 중요한 것은 여기서 중복이 제거되고, 통합된다는 것이다. 이 모델을 바탕으로 논리 모델로 변환하게 된다.



<그림 1.8.2>

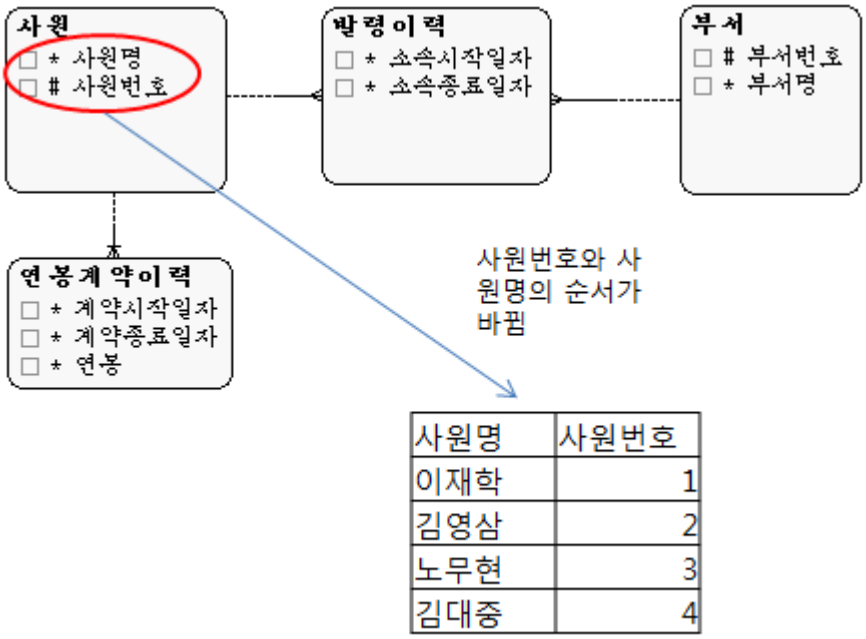
논리 모델은 곧바로 DBMS에 반영하게 된다. DBMS에 반영될 때는 디스크 배열, 인덱스 등의 물리적인 접근 방법이나 처리 방법에 대해 고려하게 된다. 종합적으로 살펴보면 다음과 같은 모델이 만들어져야 할 것이다.



<그림 1.8.3>

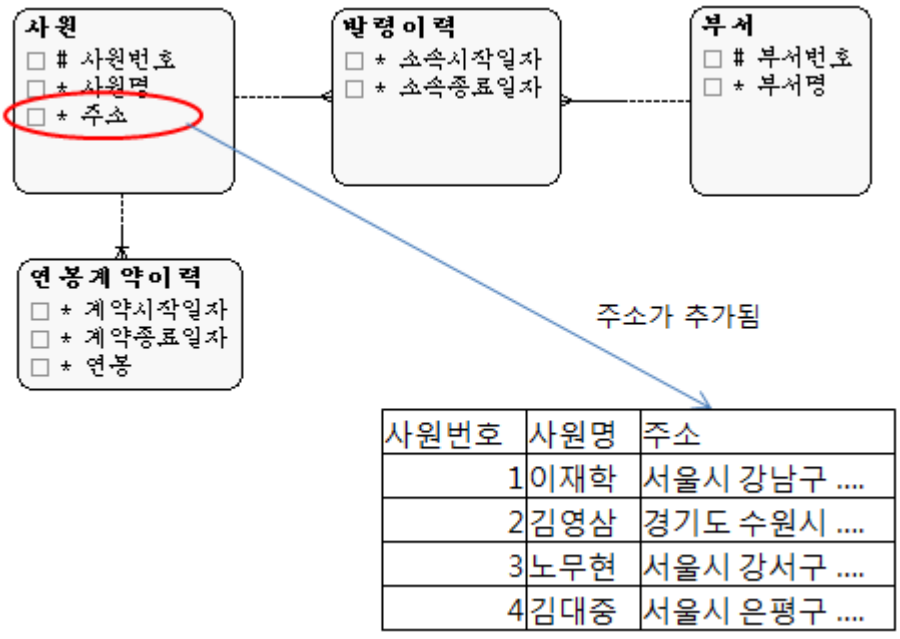
삼단계 스키마 구조는 데이터베이스를 바라보는 관점으로 각각을 분리한 것이다. 분리함으로써 얻는 이점은 '중복의 배제'와 '독립성(약한 결합도)'이다. 데이터의 중복은 개념 단계에서 외부 사용자들의 관점을 통합함으로써 얻는 이점이다. 데이터의 독립성은 논리적 독립성과 물리적 독립성이 있는데, 이는 파일 시스템에서 데이터베이스 시스템으로 바뀌는 역사적인 배경으로써도 한몫 하는 이점이다.

논리적 독립성은 개념스키마가 논리적으로 변경이 되어도 각 사용자의 전체의 관점을 포함하고 있다면 외부스키마에 영향을 끼치지 않음을 의미한다. 예를 들어, 개념 스키마의 사원번호와 사원명의 순서가 바뀌었다고 가정해 보자.



<그림 1.8.4>

만약 이렇게 변경되었을 때에 개발된 프로그램 소스의 유지보수가 필요하다면 릴레이션형 데이터베이스를 제대로 활용하지 못하고 있는 것(릴레이션의 특징 중 어트리뷰트의 무순서 위반)이다. 이는 외부단계와 개념단계가 강한 종속성이 있는 것이다. 또 다른 예로 사원의 주소가 필요한 외부 사용자의 관점이 새롭게 만들어졌을 경우 개념 스키마는 다음과 같이 변경될 것이다.



<그림 1.8.5>

새롭게 주소 속성이 추가된 경우에라도 외부 사용자 관점이 영향을 받아서는 안 된다. 역시 이 경우에도 어떤 문제가 발생한다면 데이터베이스를 제대로 활용하지 못하고 있는 것이다.

물리적 독립성은 물리적으로 어떤 변경이 있어도 개념스키마와 외부스키마에 영향을 끼치지 않음을 의미한다. 저장 장치의 저장 공간이 더 필요하여 디스크를 추가하고 데이터를 새롭게 추가한 디스크에 저장한다 해도 외부 사용자의 관점에서는 전혀 문제가 없다. 이것이 물리적 독립성이다.

독립성은 업무 변화에 따른 정보시스템의 변경을 유연하게 해준다. 즉, '유지보수가 쉬워진다'로 압축할 수 있다. 유지보수가 쉽다는 것은 개발의 편의성도 제공해준다는 의미와 같다. 물론 데이터베이스 시스템을 파일 시스템의 방식으로 사용하면 독립성이 가지는 진정한 의미를 찾을 수 없다. 설계 자체를 데이터베이스 시스템에 맞게 하는 것이 중요하다.

1.9 데이터베이스 시스템

데이터베이스 시스템을 크게 2가지 관점에서 바라 볼 수 있다.

- 데이터 모델 관점
- 데이터베이스 관리 시스템(DBMS) 관점

관점이 2가지가 있지만 모델 관점에서 데이터베이스 시스템을 바라보는 것이 중요하다. 왜냐하면 우리는 DBMS를 직접 개발하지 않기 때문이다. Oracle사의 Oracle, Microsoft사의 SQL Server, IBM의 DB2 등 이미 여러 상용 DBMS 제품들이 있다. 이 제품들은 데이터베이스 이론을 토대로 만들어졌고, 우리는 데이터베이스 구축을 위하여 상용 DBMS를 구매하여 사용한다. 그러므로 데이터베이스 관리 시스템(DBMS) 관점은 DBMS 제품에 대한 공부를 하면서 충분히 살펴볼 수 있다.

데이터 모델 관점

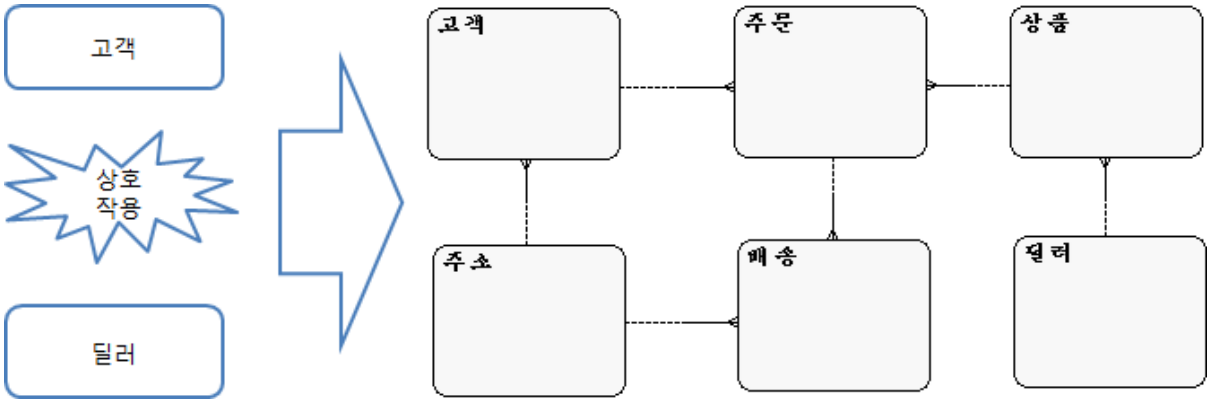
앞서 데이터베이스가 무엇인지 알아 보았으니, '시스템'이 무엇 인지만 알면 이 글의 주제인 '데이터베이스 시스템'을 알 수 있다. 시스템(system)을 이해하기 위해서는 좋은 예가 필요한데 필자가 보기에 가장 좋은 예가 '시너지(synergy)'라는 용어다. 시너지는

synergy = system + energy

의 합성어다. 다음과 같은 문장도 많이 접했을 것이다.

A사와 B사의 합병으로 시너지가 기대된다.

즉, 시너지는 +α (플러스 알파)다. 실제로 네이트온, SK텔레콤이 뭉쳐서 엄청난 시너지를 냈다. 그 덕에 네이트온은 msn메신저를 누르고 점유율 1위를 할 수 있었다. 만약 무료문자 100통이 없었다면 msn메신저를 능가할 수 있었을까? 아마도 어려웠으리라 생각된다. 데이터베이스 시스템도 마찬가지다. 여러 개체들이 모여 집합을 이루고, 이질적인 집합과 집합 사이에 상호작용으로 많은 일을 할 수 있다. 아래의 <그림 1.9.1>을 보자.

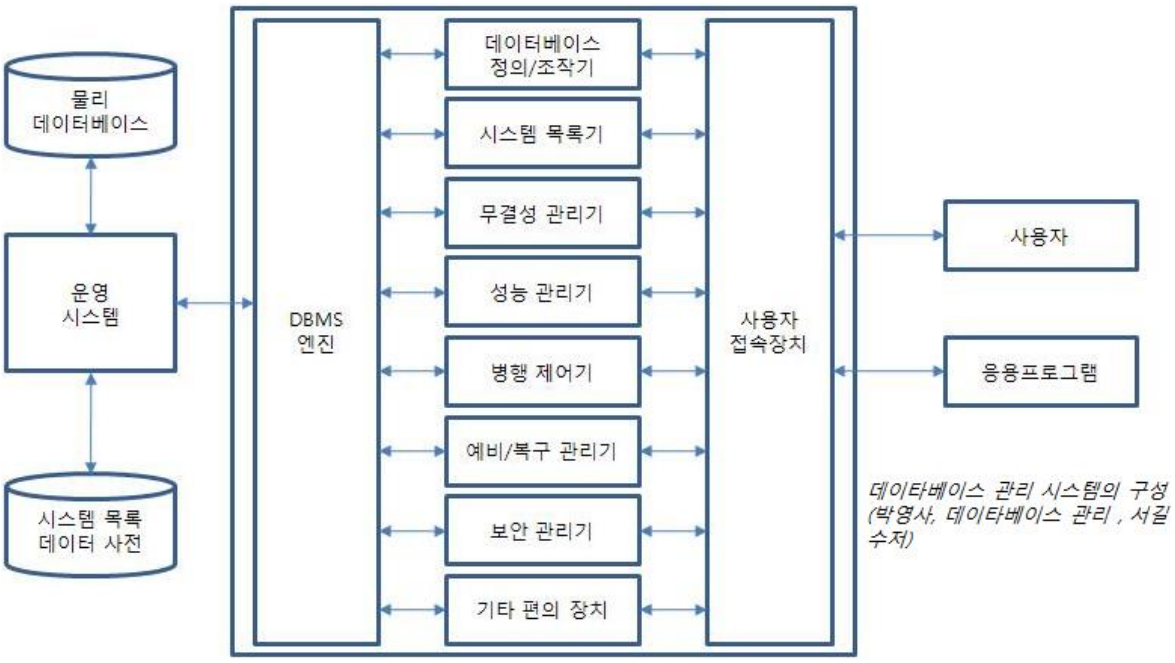


<그림 1.9.1>

고객과 딜러가 있어야만 주문도 하고 배송도 된다. 고객과 상품이 관계를 가짐으로 뭔가가 더 생산되었다. 이것이 고객과 딜러의 상호작용에 의해 발생한 시스템 에너지 즉, 시너지다. 데이터베이스 시스템은 각각의 개체 고유의 에너지와 개체와 개체간의 상호작용에 의해 시스템 에너지가 만들어(created) 진다.

데이터베이스 관리 시스템 관점

데이터베이스 시스템의 구성요소는 대략 다음 그림과 같다. 이러한 구성요소들이 상호작용을 하면서 데이터베이스 시스템을 유지한다. 구성요소 하나가 문제가 생긴다면 원래 의도하는 기능을 제대로 수행하기 어려울 것이다.



<그림 1.9.2>

복잡성

데이터 모델 관점에서의 복잡성은 개체 수와 개체들의 상호작용 횟수에 의해 결정된다. '복잡하다'와 '복잡하지 않다'의 기준은 해당 조직의 IQ에 따라 다를 것이다. 조직의 IQ가 높다고 시스템의 생산성이 높은 것도 아니다. 오히려 복잡성에 의해 생산성이 저하되고, 이익이 감소될 수 있다.

DBMS 제품의 관점에서 보면 단순하기란 쉽지 않다. 왜냐하면 경쟁을 해야 하기 때문이다. 기능의 복잡성과 DBMS 제품에서 쏟아내는 지식의 양과 난이도 및 속도는 어떤 DBMS제품이건 거의 비슷한 수준이고, 제어권도 우리에겐 없으므로 주요 기능을 위주로 해서 잘 학습하는 것이 우리가 할 수 있는 유일한 방법일 것이다.

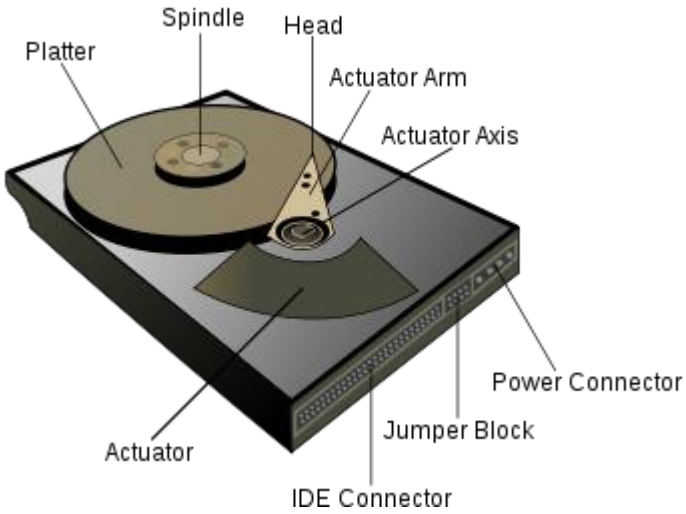
1.10 Disk와 RAID

요즘은 데스크탑이나 노트북에서 SSD(Solid-State Drive)를 주 저장 장치로 많이 사용하고 있는 추세다. 또한 SSD는 서버용 저장 장치로도 많이 사용되고 있어 OLTP 환경에서는 성능문제를 많이 해결하는 역할을 하고 있다. Hard Disk(이하 Disk)는 이제 자기테이프를 대신할 대용량의 느린 저장 장치가 될 것이고, SSD가 Disk 자리를 차지할 것이다. 하지만, 현재는 대부분 Disk가 사용되고 있고, 하드웨어의 원가가 0원이 되기 까지는 3~5년이란 시간이 걸릴 것이다. 현재 빅데이터 솔루션으로 유일하다고 할 수 있는 Hadoop은 저사양의 하드웨어를 지향하고 있다. 그러므로 Disk에 대한 학습은 여전히 가치가 있다.

이번 절에서는 Disk의 구조를 살펴보고, 그 구조로 인한 한계점을 인식하여 Disk를 어떻게 하면 제대로 활용할 수 있는지에 대해서 살펴볼 것이다. 또한 고장이 잘 나는 Disk의 가용성을 높이는 방법과 스토리지 네트워크에 대한 기본적인 개념도 살펴볼 것이다.

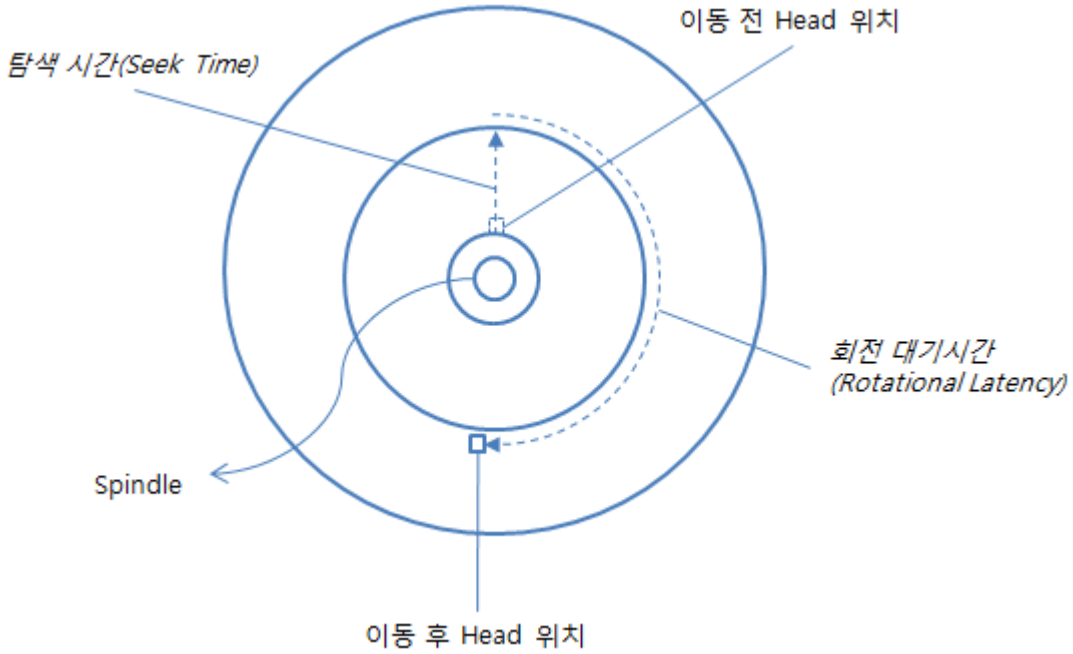
Disk 성능

컴퓨터를 구성하는 부품 중에서 디스크는 고장 확률이 가장 높은 구조를 가지고 있다. 다음은 Disk의 구조다.



<그림 1.10.1>http://en.wikipedia.org/wiki/File:Hard_drive-en.svg

Disk는 컴퓨터를 구성하는 부품 중 물리적인 움직임이 가장 크다. Spindle과 Arm이 움직여서 데이터를 찾아낸다. 한 쪽 방향으로만 움직이므로 정말 운이 없다면 360도 회전을 해야 원하는 데이터를 찾을 수 있을 것이다. 또한 Head가 Platter의 중심에서 바깥쪽까지 이동하는 시간도 데이터를 찾는 시간에 포함된다.



<그림 1.10.2>

<그림 1.10.2>은 데이터를 탐색하는데 필요한 지연시간을 나타내며, Disk에서 데이터 입/출력에 필요한 시간은 탐색시간과 회전 대기시간의 합이 된다.

I/O time = Seek Time + Rotational Latency

그러므로 10,000 RPM(Revolution Per Minute) 은 1분에 10,000번 회전을 의미하므로 초당 167

(10,000/60)회의 회전을 한다. 1회전에 걸리는 시간은 6ms(1분 : 10,000회전 = x : 1회전) 이다. 그러므로 3.0 ms(6ms/2)의 평균 회전 대기시간을 가짐을 알 수 있다.

RPM(Revolution Per Minute)	Max Rotational Latency(ms)	Avg Rotational Latency(ms)
4200	14.3	7.15
5400	11.1	5.65
7200	8.3	4.15
10000	6	3
15000	4	2

<표 1.10.1>

탐색 시간(Seek time)은 디스크마다 조금씩 다르지만 제품마다 평균 탐색 시간(Avg Seek Time)을 제공하므로 하드웨어에서 제공하는 문서를 참고하면 된다. 참고로 다음의 URL은 평균 탐색 시간으로 Disk를 검색할 수 있다.

http://reviews.cnet.com/hard-drives/?sa=500078

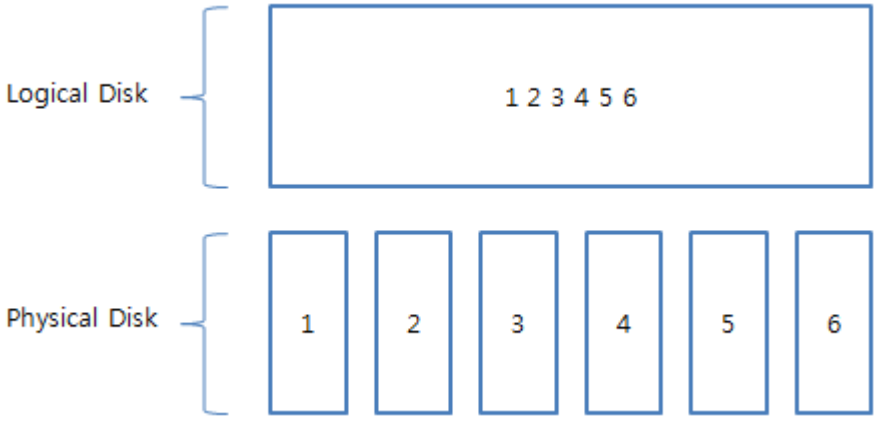
만약 Disk가 10,000 RPM이고, 평균 탐색 시간이 4ms라면 Disk는 한 번의 I/O를 위해서 약 7ms(3ms + 4ms)의 시간이 소요됨을 알 수 있다. 그러므로 1초에 약 143(=1,000 / 7)회의 I/O가 가능하다. 이것이 흔히들 이야기하는 IOPS(Input Output Operation Per Second)다. 여기서는 143 IOPS다. IOPS는 Disk 성능을 측정하는 하나의 단위이므로 위의 계산을 참고하여 구매 시 참고하면 된다. 중요한 것은 비용(돈)이다. 돈이 많으면 좋은 부품을 구매하라. 하드웨어는 투자한 만큼의 효과가 있다.

RAID(Redundant Array of Inexpensive (or Independent) Disks)

RAID 시스템은 쉽게 말해 Disk의 배열(Array)이다. 논리적인 Disk가 아닌 물리적인 여러 개의 Disk를 하나의 Disk처럼 사용하는 것이다. 여러 Disk를 하나의 Disk처럼 사용하므로 다수의 디스크에 데이터를 분할하여 병렬로 전송함으로써 디스크의 입/출력을 향상시킨다. 자연스럽게 성능상의 이점이 생기지만 구성에 따라서는 성능의 트레이드 오프(Trade-Off)가 발생하기도 한다.

Disk는 컴퓨터를 구성하는 부품 중에 가장 고장이 잘나는 부품이다. RAID는 고장에 대비하여 데이터를 보호하고, 가용성을 늘리는 기술이다. 물론 RAID 구성 방법이나 Disk 고장 패턴에 따라서 Disk 고장에 대한 손실도 막을 수도 있고, 데이터가 유실될 수도 있다.

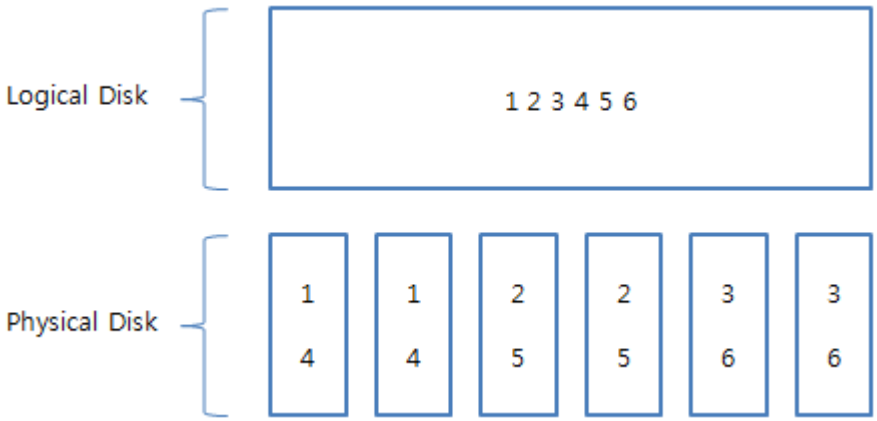
RAID는 구성 형태에 따라서 여러 레벨로 나뉘어 지는데, RAID-0, RAID-1, RAID-5만 알면 된다. RAID-2, RAID-3, RAID4는 실무에서 쓰이지 않는다. RAID-1+0, RAID-0+1은 RAID-0과 RAID-1의 결합으로 표준을 약간 변경시킨 것으로 공시적인 RAID 레벨로 인정하지는 않는다고 한다. RAID-0은 <그림 1.10.3>과 같다. {1,2,3,4,5,6}은 저장된 데이터다.



<그림 1.10.3>

각 디스크들은 훨씬 작은 쓰기 작업을 동시에 끝낼 수 있으므로 저장 속도는 싱글 디스크에 비해 빠르다. 또한 분리된 컨트롤러 사이에 저장할 경우는 성능이 더욱 향상된다. 하지만 RAID-0은 가용성을 감소시킨다. 디스크 중 1개만 고장이 나도 전체 스트라이프는 동작하지 않기 때문이다. 디스크의 고장 확률만큼 더욱 가용성을 떨어뜨린다. 그러므로 실제 서비스로는 거의 사용되지 않는다.

<그림 1.10.4>는 RAID-1이다. RAID-1은 두 번째 디스크에 첫 번째 디스크의 모든 바이트를 복사해서 유지한다. 100% 동기화되며, 한 쪽 디스크가 고장이라도 다른 디스크가 문제없이 계속 동작한다.

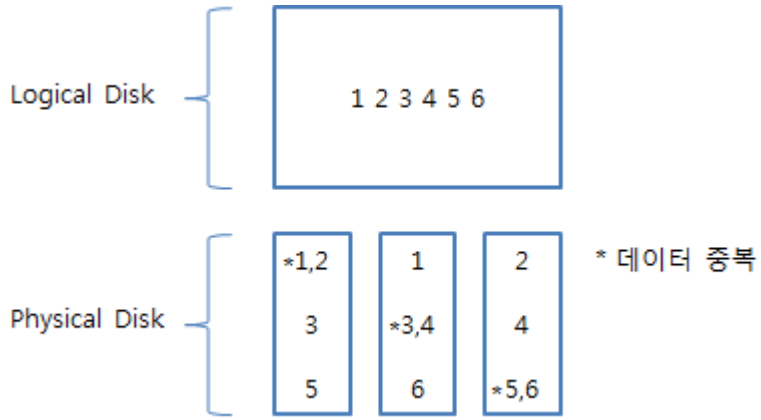


<그림 1.10.4>

RAID-1의 가장 큰 장점은 디스크 고장으로 인한 데이터 손실을 막는 것이다. 또한 병렬로 읽기를 요청하여 두 디스크 중에 먼저 응답한 디스크를 사용하게 되어 읽기 성능이 싱글 디스크에 비해 향상된다. 가장 큰 단점으로는 비용이 2배로 증가(각 미러는 100% 디스크 오버헤드)하는 것이다. 고장 났거나 새 RAID-1의 재동기화는 원본 디스크의 내용의 블럭 대 블럭의 완벽한 복사 후에나 가능하다. 이 때 엄청난 I/O 발생으로 부하가 발생한다. 한 쪽 디스크의 쓰기 작업이 끝나지 않았거나 디스크의 성능 차가 발생하면 쓰기 성능이 저하되며, 싱글 디스크보다 느리다. 하지만 쓰기 캐시를 조정함으로써 어느 정도 부하를 조정할 수 있다.

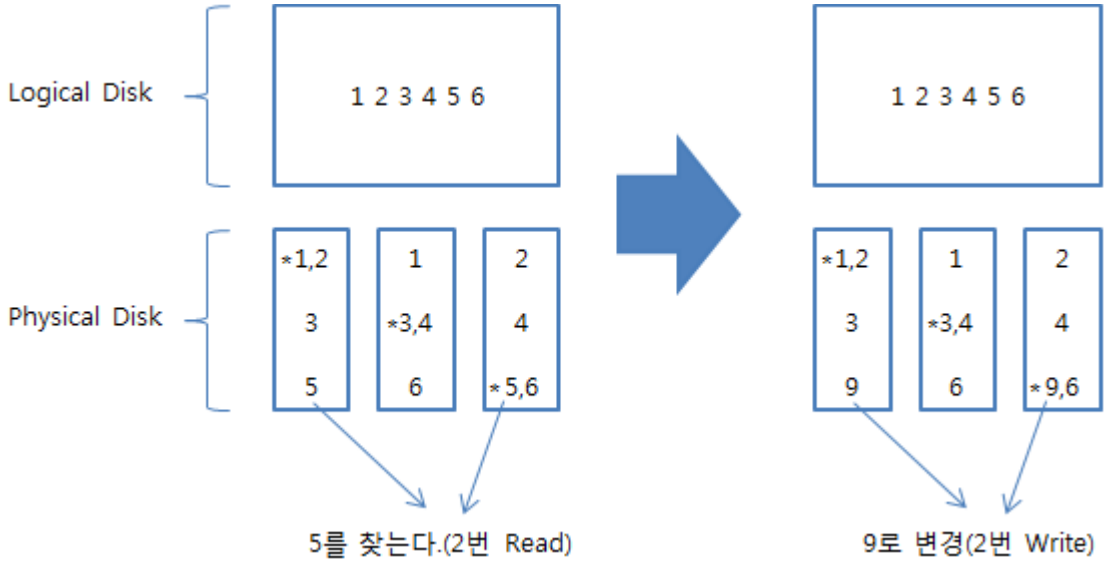
RAID-1은 Master/Slave 관계가 아니다. 즉, 한 쪽 디스크가 고장 나면 다른 디스크가 고장 난 디스크의 동작을 이어 받는 것이 아니다. 즉, Fail Over가 아니다. 백업이 필요 없다는 것도 RAID-1에 대한 잘못된 생각이다. 만약 데이터를 실수로 삭제한다면 데이터를 복원할 필요가 있다.

<그림 1.10.5>는 RAID-5를 나타낸 그림이다. 쓰기 작업이 싱글 디스크에 비해 15 ~ 20% 정도 느리기 때문에 성능 문제가 발생한다는 사실을 알아야 한다.



<그림 1.10.5>

RAID-5는 한 번의 쓰기에서 다음과 같이 4번의 I/O가 발생한다. 예를 들어 5를 9로 변경한다고 가정해 보자. 그러면 먼저 변경해야 할 데이터를 읽어야 하는데, 중복 저장된 데이터까지 읽어야 하므로 2번의 읽기가 발생한다. 그리고 5를 9로 변경하기 위해 데이터와 중복 저장된 데이터에 대해 모두 변경해야 하므로 2번의 쓰기가 발생한다. 그러므로 RAID-5는 한 번의 쓰기에 대해 4번의 I/O가 발생한다.



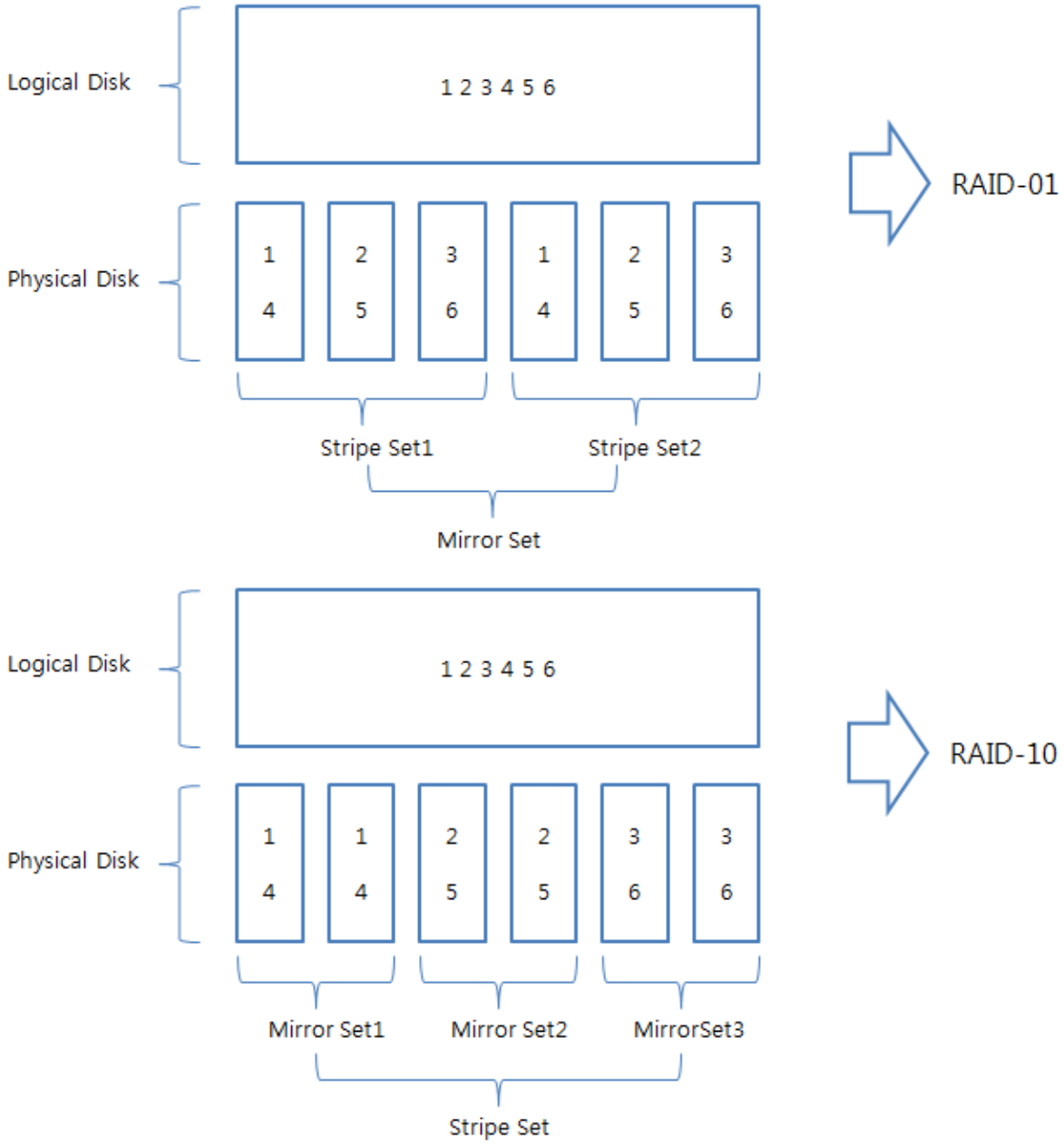
<그림 1.10.6>

RAID-5는 RAID-5로 묶인 디스크 중에 1개의 고장에 대해서만 데이터의 손실을 막아준다. <그림 1.10.5>에서 첫 번째 디스크가 고장이 났다면 두 번째, 세 번째 디스크의 중복 저장된 데이터가

있으므로 데이터는 보존된다. 하지만 디스크가 2개 이상 고장 난다면 모든 데이터는 손실된다. RAID-6은 중복 저장된 데이터를 1개 더 두는 방식이다. 즉, 중복 저장된 데이터가 2개이므로 디스크가 동시에 3개 이상 고장 나지 않는다면 데이터를 보존할 수 있다.

일반적으로 디스크의 병목이 있다고 판단되고 쓰기 비율이 20%를 초과하면 RAID-5가 적당한 것인지 고민을 해봐야 한다. 디스크가 고장 난 경우도 고장 복구를 위해서 고장 나지 않은 스트라이프된 모든 디스크를 모두 읽어서 새로운 디스크에 데이터를 복구해야 하므로 디스크 손실을 복구하는데 많은 시간이 소모된다. 또한 디스크가 복구 전에 1개 이상 더 고장 날 경우는 전체 데이터를 손실하게 된다.

<그림 1.10.7>은 RAID-01과 RAID-10을 나타낸다.



<그림 1.10.7>

RAID를 구성하는 순서는 각각 다음과 같다.

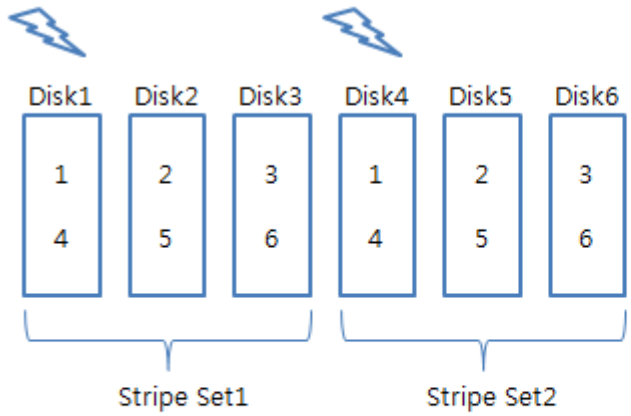
RAID-01

1. 디스크를 선택한다.
2. 디스크들끼리 스트라이프한다.
3. 새로운 디스크를 선택한다.
4. 새롭게 선택한 디스크를 스트라이프한다.
5. 스트라이프된 디스크들끼리 미러링한다.

RAID-10

1. 디스크를 선택한다.
2. 선택된 디스크들끼리 미러링한다.
3. 미러된 디스크들을 스트라이프한다.

RAID-01과 RAID-10은 고장 날 경우 데이터 손실을 막을 수 있는 확률이 다르다. RAID-01의 경우 Stripe Set1의 디스크 중1개의 디스크가 고장 날 경우 Stripe Set2에 데이터가 미러되어 있으므로 데이터의 손실을 막을 수 있다. 하지만 Stripe Set2의 디스크가 추가적으로 고장 날 경우는 모든 데이터가 손실 된다. 다음의 <그림 1.10.8>은 RAID-01이 어떤 경우에 모든 데이터가 손실되는지를 보여준다.

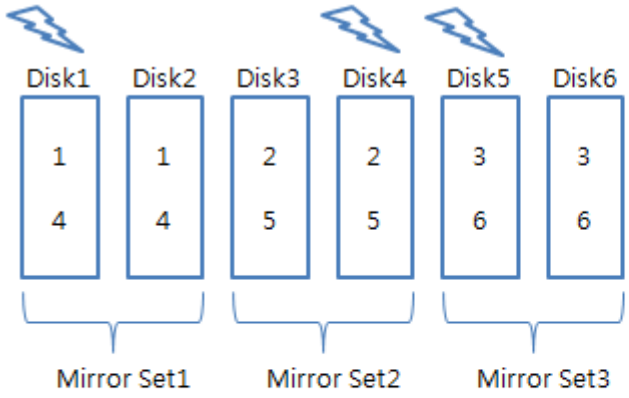


1. Disk1이 고장나면
2. Stripe Set1의 데이터는 모두 손실되지만 Stripe Set2가 있으므로 데이터는 보존
3. 이 때, 추가적으로 Disk4가 고장나면
4. Stripe Set2의 데이터가 손실되어 모든 데이터 손실

<그림 1.10.8>

RAID-10의 경우에 Mirror Set1의 디스크 중1개의 디스크가 고장 날 경우 미러된 데이터가 있으므로 데이터의 손실을 막을 수 있다. Mirror Set2, Mirror Set3도 마찬가지다. 하지만 미러된 디스크도 고장이 난다면 모든 데이터가 손실된다. 다음의 <그림 1.10.9>은 RAID-10이 어떤 경우에 모든 데

이터가 손실되는지를 보여준다.



1. Disk1이 고장나면
2. Disk1의 데이터는 모두 손실되지만 Disk2의 미러된 데이터가 있으므로 데이터는 보존
3. 이 때, 추가적으로 Disk4가 고장나면
4. Disk4의 데이터는 모두 손실되지만 Disk5의 미러된 데이터가 있으므로 데이터는 보존
5. 이 때, 추가적으로 Disk5가 고장나면
6. 모든 데이터는 손실

<그림 1.10.9>

<그림 1.10.8>과 <그림 1.10.9>을 보면 알 수 있듯이 데이터 손실을 막을 수 있는 확률은 RAID-1+0이 훨씬 더 높다. 만약 Diks1이 고장일 때, 전체 데이터를 손실할 경우는 다음과 같다.

- RAID-01: Mirror Set1 은 사용할 수 없으므로 Disk4 ~ 6 중 하나라도 고장이 난다면 모든 데이터 손실
- RAID-10: Disk2 만 고장이 나지 않으면 데이터가 손실되지 않는다.
- 데이터 손실 확률은 RAID-01 은 60%(3/5), RAID-10 은 20%(1/5)가 된다.

또 다른 문제는 RAID-01의 경우 Disk1이 고장인 경우 Disk1, Disk2, Disk3로 구성된 전체 스트라이프를 잃게 되며, 재구성 시에Disk4, Disk5, Disk6를 모두 동기화해야 하지만, RAID-10은 단지 Disk1만 재동기하면 된다. RAID-10의 유일한 단점은 스트라이프된 각각의 구성 요소가 동일한 크기를 가질 필요가 있다는 것이다. RAID-01은 이런 사항을 요구하지 않는다. 일반적으로 RAID-1은 RAID-10으로 보면 된다.

RAID구성시 Disk의 ReadCount/Sec와 WriteCount/Sec를 알고 있다면 다음과 같은 공식을 이용하여 디스크의 I/O 횟수를 추정 할 수 있다. RAID-10과 RAID-01은 RAID-1의 변형이므로 RAID-1의 공식을 이용한다.

- RAID-0 : (Read + Write) / Disk 수
- RAID-1 : (Read + 2 * Write) / Disk 수 --> Write 할 때에 미러링하므로 2 배의 Write
- RAID-5 : (Read + 4 * Write) / Disk 수 --> 앞서 설명했듯이 Write 는 4 회의 I/O 필요

최근에 소프트웨어를 크게 강조하지만, 여전히 하드웨어의 비약적인 발전을 이용하는 것은 매우 유용하다. 특히 데이터 저장 장치는 아직도 매우 느린 하드웨어 장치이므로 이 부분에 비용을 투

자하는 것을 적극적으로 고려해야 한다.

1.11 인덱스와 해시

인덱스(Index)는 원하고자 하는 데이터만을 효율적으로 조회, 갱신, 삭제를 위해 존재한다. 인덱스가 없었다면 데이터베이스도 없었을 것이다. 그 만큼 매우 중요하다. 실무적으로는 매우 많이 활용되지만, 구현은 하지 않는다. DBMS 제품 개발을 할 것이 아니므로 인덱스를 활용함으로써 얻는 이득과 손실을 잘 따져볼 수 있을 정도의 지식만 습득하면 된다.

인덱스 개요

인덱스는 검색의 효율성을 높이기 위한 물리적인 데이터 구조다. 인덱스는 책 앞부분의 목차나 뒷부분의 찾아보기로 생각하면 된다. '목차'를 통해 개략적인 정보를 알고 해당 페이지로 이동하여 원하고자 하는 책의 내용을 찾을 수 있다. 또한 특정 단어에 대한 설명을 '찾아보기'를 통해서도 쉽게 찾을 수 있다.

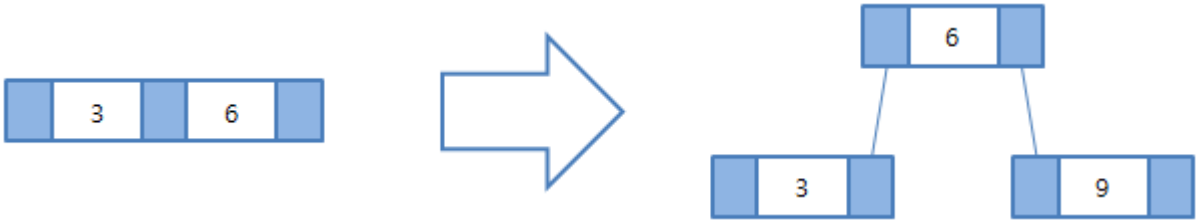
만약 찾아보기나 목차가 없었다면 우리는 책을 처음부터 끝까지 뒤져야 한다. 만약 그 키워드가 책에서 딱 한 부분에서만 설명된다는 보장이 있다고 가정하자. 운이 있으면 몇 페이지 뒤지지 않고도 해당 내용을 찾을 수 있다. 하지만 해당 키워드가 책의 여러 분산된 페이지에서 다루는 내용이라면 우리는 책을 끝까지 뒤져야 한다. 책의 목차가 다음과 같은 경우를 보자.

1. 데이터베이스 기초
2. 데이터 모델링
3. SQL
4. ...

만약 1과 2사이에 '프로세스 모델링'이라는 단원이 추가된다면 '데이터 모델링'이후부터는 모두 숫자를 바꾸어야 한다. 책의 뒤는 어떠한가? 찾아보기도 추가되어 다시 정렬해야 한다. 만약 3장 SQL이 삭제되면 어떠한가? SQL 뒤에 있는 다른 Chapter는 모두 다시 번호를 매겨야 한다. 데이터베이스의 인덱스는 이와 거의 비슷하게 데이터의 삽입, 삭제와 같은 연산에 인덱스 구조를 유지하기 위한 추가적인 비용이 필요하다. 왜 추가적인 비용이 필요한지 알아보고, 인덱스의 종류를 살펴보자.

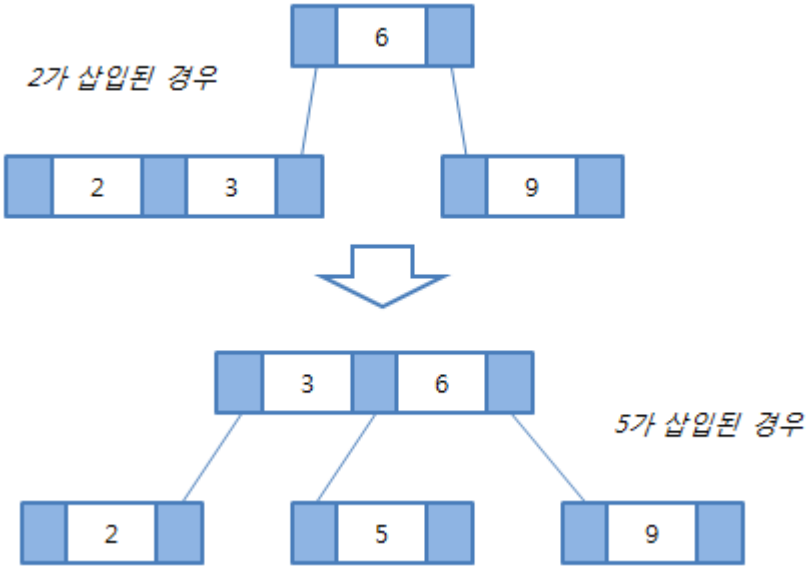
Balanced Tree Index

<그림 1.11.1>은 Binary Tree에 3과 6이 삽입된 상태에서 9가 삽입되면 어떻게 인덱스의 구조가 변경되는지를 보여준다.



<그림 1.11.1>

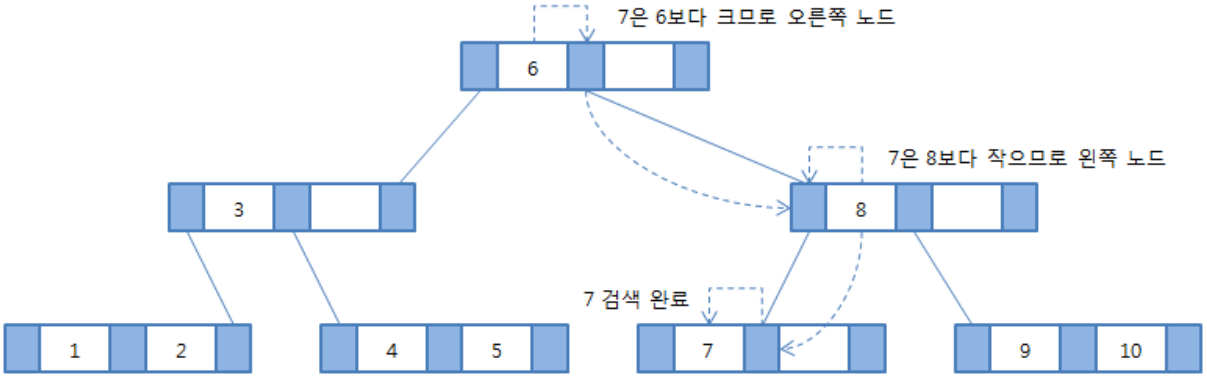
3과 6이 삽입된 상태에서는 9가 삽입될 공간이 없으므로 3보다는 크고 9보다 작은 수인 6은 상승(promotion)하고 3과 9는 분할(splitting)된다. 이 상태에서 만약 2와 5가 삽입된다면 인덱스는 <그림 1.11.2>와 같이 변경된다.



<그림 1.11.2>

2는 3보다 작고 6보다도 작으므로 3보다 작은 위치에 삽입된다. 하지만, 5는 6보다는 작고 2, 3보다는 크고, 9보다는 작다. 그러므로 리프(leaf) 노드에서 분할이 일어나고 2보다 크고 5보다 작은 3이 상승한다.

검색은 삽입보다는 이해하기 쉽다. 1 ~ 10까지 삽입된 <그림 1.11.3>에서 인덱스에서 7을 검색해 보자.

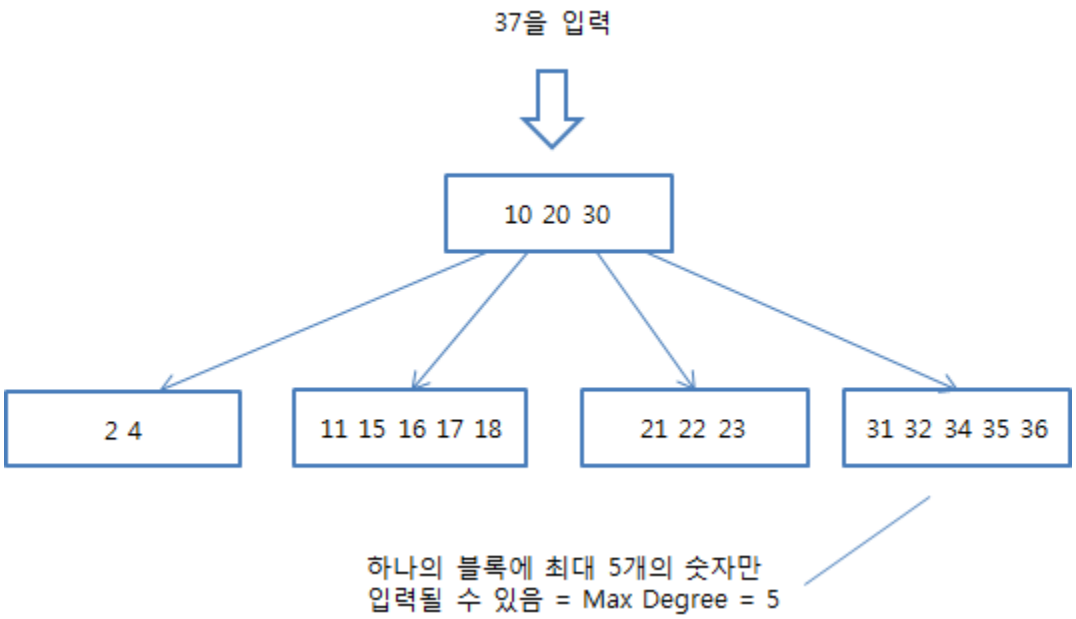


<그림 1.11.3>

1. 6과 7을 비교 6 < 7 오른쪽 노드로
2. 7과 8를 비교 7 < 8 왼쪽 노드로
3. 7 검색 완료

데이터가 많더라도 인덱스를 이용한다면 데이터를 처음부터 끝까지 찾지 않아도 된다. 인덱스는 검색을 위한 메커니즘이다. 삽입에서 살펴보았듯이 검색보다는 갱신/삭제/삽입은 많은 비용이 드는 일이다. 하지만 갱신/삭제는 원하는 적은 양의 데이터만을 변경하는 것이므로 인덱스를 이용하는 것이 인덱스를 이용하지 않을 때보다도 효과적이다. 대부분의 어플리케이션이나 사용자는 검색을 주로 하기 때문에 인덱스의 사용은 거의 필수적이라고 할 수 있다. 물론 갱신과 삭제도 검색이 일어난 후에나 가능하다.

일반적으로 DBMS에 구현된 인덱스는 Balanced Tree로 기존의 Binary Tree의 치우침(skewed)의 단점을 보완하여 데이터의 삽입, 삭제 시 Tree구조를 균형이 잡히도록 재구성한다. 데이터를 삽입하는 예제를 통해 인덱스의 구조를 유지하기 위한 동작들을 살펴보자.



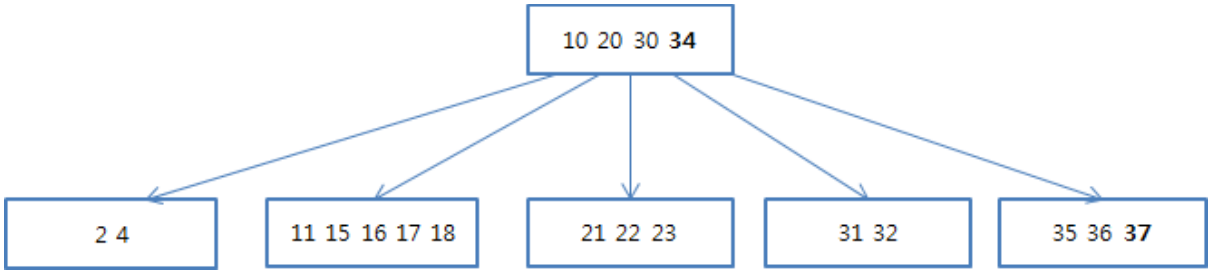
<그림 1.11.4>

<그림 1.11.4>는 Balanced Tree다. 37을 입력하면 다음과 같이 동작한다.

1. Root 노드에서 37의 위치를 찾는다.
2. 37은 30보다 크므로 가장 오른쪽의 블록에 삽입되어야 한다. 하지만, 블록이 꽉 차서 더 이상 삽입될 공간이 없어 분할해야 한다.
3. 가장 오른쪽의 블록의 중간값을 기준으로 블록을 분할 한다.

4. Root노드에 중간값인 34가 입력되고, 가장 오른쪽 블록에 37을 삽입한다.

데이터 삽입 결과는 <그림 1.11.5>와 같다.

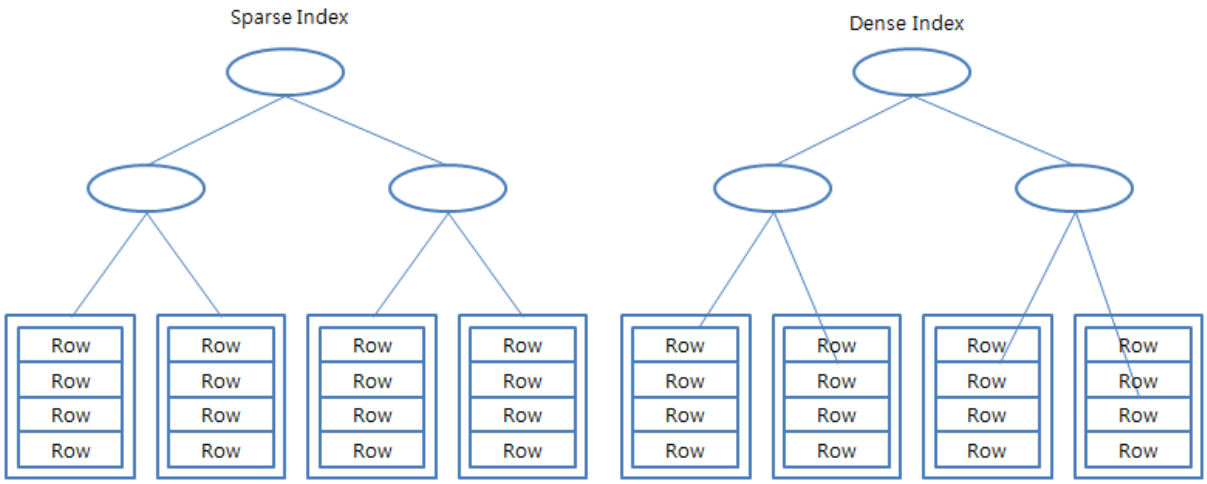


<그림 1.11.5>

Binary Tree와 Balanced Tree에 모두 데이터가 삽입될 때에 대한 예제를 살펴보았다. 실제로 우리가 Tree를 구현할 것이 아니므로 알고리즘을 자세히 알 필요는 없다. 그러나 굳이 이렇게 살펴보는 이유는 한 번만 알고리즘을 들여다 보면 인덱스 이용 시 왜 데이터가 빠르게 검색되는지, 데이터의 변경에 부하가 발생하는지 알 수 있기 때문이다.

Sparse Index, Dense Index

인덱스의 리프 노드(leaf node)가 페이지(page) 또는 블록(block)을 가리키고 있으면 Sparse Index다. 리프 노드가 특정한 행(row)을 가리키고 있다면 Dense Index다. 아래의 <그림 1.11.6>는 Sparse Index와 Dense Index의 구조를 나타낸다.



<그림 1.11.6>

페이지에는 여러 개의 Row가 있으므로 일반적인 경우는 Sparse Index가 유리하다. 하지만 레코드의 길이가 페이지의 크기와 비슷한 경우라면 Sparse Index는 Dense Index보다 디스크 액세스를 많이 수행하게 되므로 성능이 떨어진다.

상용 DBMS제품은 다음과 같은 종류를 인덱스를 가지고 있다. Oracle에서는 인덱스 조직 테이블(Indexed-Organized Table)이라고 해서 MSSQL Server의 클러스터드 인덱스를 가진 테이블과 비슷한 구조로 생각하면 되는데, 이는 Sparse Index 방식이라고 보면 된다.

DBMS	Primary Data Structures	Secondary Data Structures
IBM DB2 UDB	B-Tree(D)	B-Tree
Oracle	B-Tree(D), Hash(D)	B-Tree, Bitmap, Function
MSSQL	B-Tree(S)	B-Tree
Sybase	B-Tree(S)	B-Tree

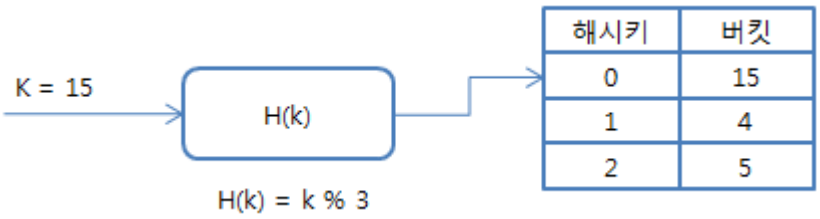
* D: Dense Index , S: Sparse Index
* Secondary Index는 무조건 Dense Index 임

또 한 가지 중요한 사실은 Sparse Index (Primary Index)와 Dense Index (Secondary Index)가 혼재되어 있을 경우 Disk에 따로 나누어 저장하는 것인 효과적이라는 것이다.

해시

해시(hash)는 행의 위치를 결정하고 특정 행이 어디에 위치하는지를 알아내는 가장 빠른 방법이다. 특정 행이 어디에 있는지는 해시 함수의 결과로 알아낸다. Oracle은 소수(1과 자신으로만 나누어 떨어지는 수)를 이용해서 행의 위치를 결정하고 알아내는데 사용한다고 한다. 해시는 DBMS마다 제공하는 것은 아니나 중대형 급의 거의 모든 DBMS 제품은 해시 기법을 제공한다.

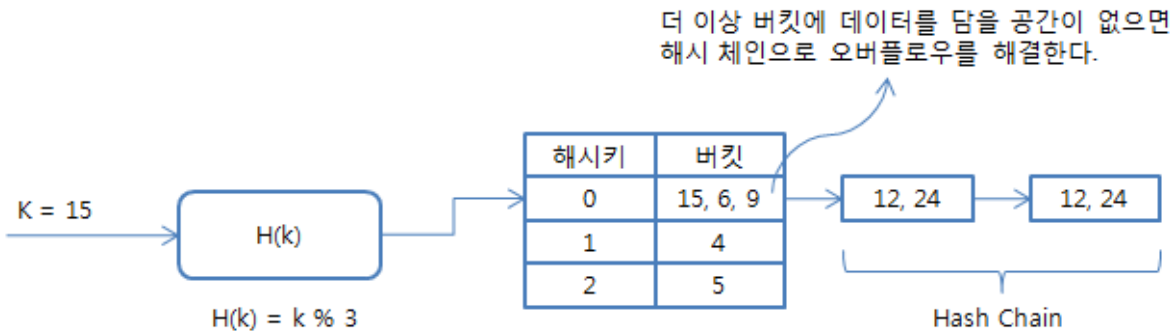
해시는 이용할 만한 가치가 충분하다. 어떠한 면에서는 해시와 인덱스가 유사한 면을 보이고 있다. 그러나 해시에서 사용되는 해시키는 데이터 행의 위치와는 직접적인 관련성이 전혀 없다. 또한 해시의 가장 큰 문제점은 하나의 해시키가 두 개 이상의 행을 가리킬 수 있다는데 문제가 있다. 그래서 유일하고 빠른 해시함수를 만드는 것이 가장 핵심일 것이다. <그림 1.11.7>는 해시의 기본 개념을 나타낸다. 여기서 해시함수는 모듈로(나머지) 연산을 하였다.



<그림 1.11.7>

그림과 같이 데이터를 해시 함수에 입력하고 그 결과로 데이터 행의 주소를 결정하는 것이다. 해시 함수는 제품마다 다르며 어떤 알고리즘을 해시 함수에 적용하는가에 따라서 해시의 성능이 좌우되기도 한다. 해시 기법은 개별적인 행의 접근에 상당히 빠른 성능을 나타낸다. 하지만 삽입, 삭제, 갱신이 자주 일어나는 컬럼에는 사용하지 말아야 한다. 만약 해시기법이 적용된 컬럼의 데

이터가 갱신되어 해시키가 변경되면DBMS는 그 행을 삭제하고 새로운 주소에 그 행을 재배치 시켜야 하기 때문에 이러한 경우는 비효율적이 될 수 밖에 없다. 또한 데이터가 가지는 값의 다양성도 고려를 해봐야 한다. 만약 "성별" 이라는 컬럼에 해시를 설정한다면 긴 체인이 생기게 된다. 그러므로 해시의 적용은 선택도가 좋은 컬럼에 적용하는 것이 효과적이다. <그림 1.11.8>은 해시 체인을 도식한 것이다.



<그림 1.11.8>

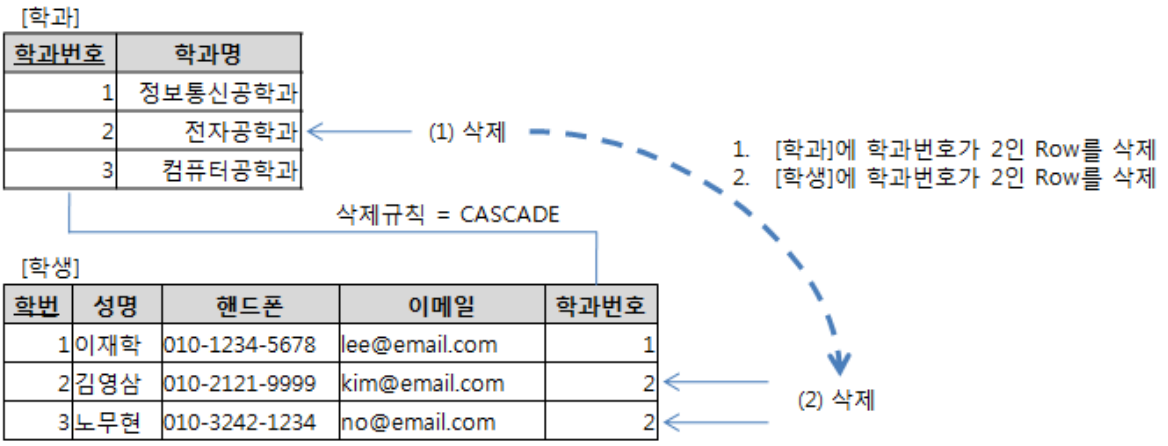
1.12 트랜잭션

많은 사람들의 오해 중에 하나는 트랜잭션의 제어권이 서버에 있는 줄 아는 것이다. 트랜잭션은 99% DBMS 사용자(어플리케이션 포함)에게 제어권이 있다. DBMS는 단지 설정된 데로 시키는 데로 움직여줄 뿐이다. 대규모 시스템에서 꼬인 트랜잭션은 블록킹과 데드락을 유발하여 시스템의 성능 저하의 원인이 된다. 특히, 비관적 동시성 제어 모델이 기본설정인 DBMS에서 블록킹이나 데드락에 의한 성능저하 현상을 경험한 사람들은 자신들의 지식이 부족함을 느끼지 못하고DBMS 제품을 탓하기도 한다.

트랜잭션은 4가지 속성이 있는데, 4가지를 모두 만족해야만 트랜잭션이라고 할 수 있다. DBMS 제품들은 트랜잭션의 4가지 속성을 만족시켜 DBMS를 구현한다. 일반적으로 2가지는 DBMS에서 보장해주고 나머지 2가지는 사용자가 제어권을 가진다.1.10절에서는 트랜잭션의 기본 개념과 동시성 제어모델, 복구에 대한 개념을 살펴볼 것이다.

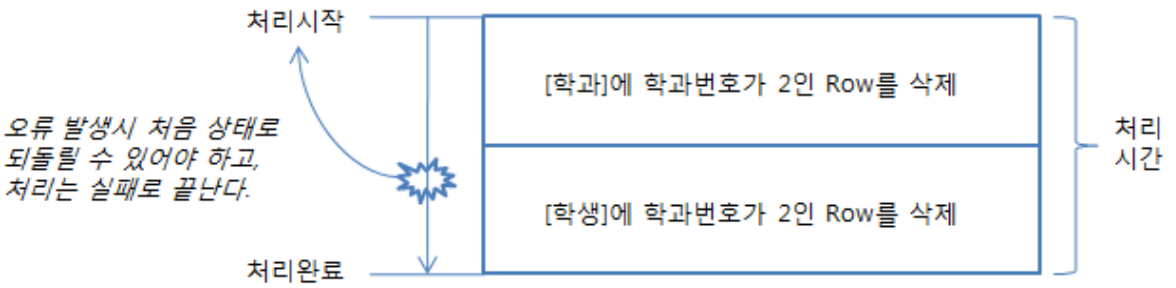
트랜잭션의 개념

트랜잭션은 데이터의 처리 단위(묶음)다. 예를 들어 다음의 <그림 1.12.1>과 같은 테이블이 DBMS에 구현되었다고 가정해야 보자. 학과번호가 2인 Row들을 [학과]와 [학생] 테이블에서 삭제하는 것이 하나의 트랜잭션이다.



<그림 1.12.1>

[학과] 테이블의 학과번호가 2인 Row를 삭제한다면 CASCADE참조 무결성 규칙에 의해 [학생] 테이블의 학과번호가 2인 Row들까지도 삭제되어야만 데이터의 처리 작업이 완료한 것이다. 하지만 어떤 사유로 인하여 [학과] 테이블만 Row가 삭제되고 [학생] 테이블의 Row를 삭제하는 중에 처리가 멈추었다면 삭제되었던 [학과] 테이블의 Row들은 트랜잭션 시작 이전 상태로 복구되어야 한다.



<그림 1.12.2>

만약 [학생] 테이블의 삭제 작업이 어떤 사유로 인하여 더디게 진행된다면 성능 문제가 발생할 수 있다. 처리 시간이 길어진다는 것은 시스템의 자원을 오랫동안 점유하여 다른 어플리케이션이나 트랜잭션이 자원을 사용할 수 없음을 뜻한다. 또한, DBMS는 데이터의 일관성을 위하여 변경 이전 버전의 데이터를 읽을 수 있도록 하거나 또는 아예 읽지 못하도록 막을 것이다. 여러 사용자가 같은 데이터에 대한 동시 다발적인 변경을 요구하는 패턴이라면 트랜잭션의 처리가 완료될 때까지 데이터의 변경을 막을 것이다. 이러한 문제를 '동시성 문제'라고 하는데, 트랜잭션의 처리 시간을 최대한 짧게 가져가게끔 처음부터 트랜잭션에 대한 설계를 잘 해야 한다.

용어

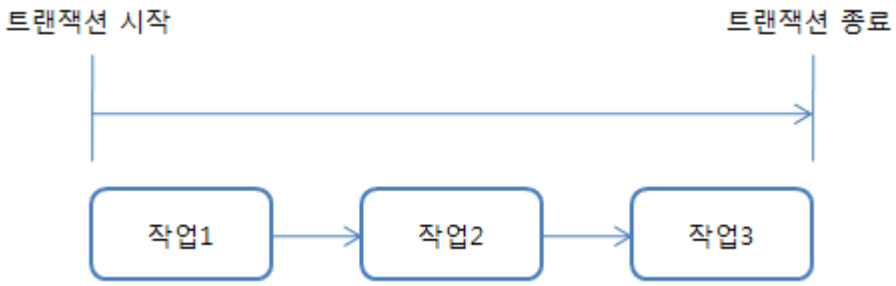
- Commit - 트랜잭션이 완료됨.
- Rollback - 트랜잭션이 실패하여 트랜잭션 시작 이전의 상태로 되돌림.

ACID

트랜잭션은 데이터의 무결성을 보장하기 위하여 다음의 4가지를 지원해야 한다.

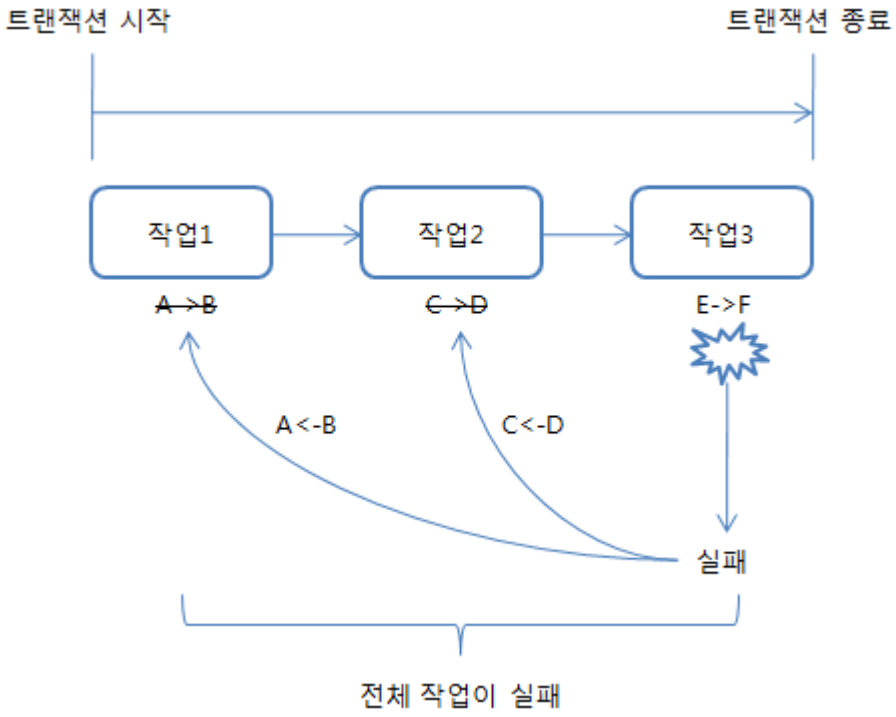
- 원자성(Atomicity)
- 일관성(Consistency) -> 사용자의 몫
- 고립성(Isolation) -> 사용자의 몫(기본적인 설정으로 일부 제어)
- 영속성(Durability)

원자성은 'All or Nothing'으로 표현할 수 있다. 원자성은 DBMS에서 구현되어 관리 된다. 다음의 <그림 1.12.3>과 같이 트랜잭션의 시작과 종료 사이의 여러 작업 중에 하나의 작업이라도 실패한다면 작업1, 작업2, 작업3 모두 실패로 끝남을 의미한다.



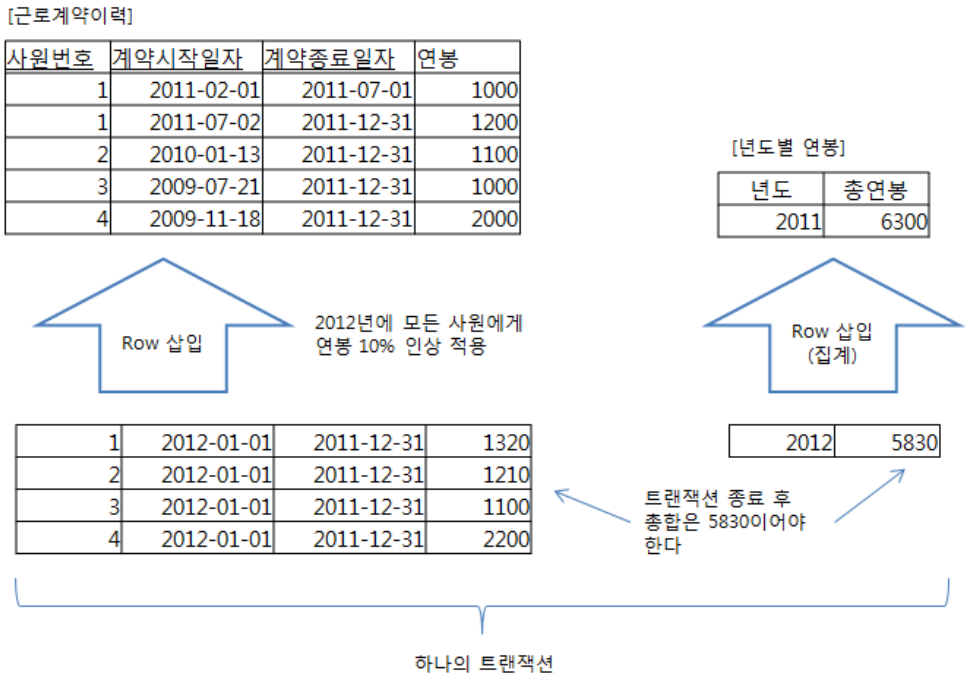
<그림 1.12.3>

예를 들어 <그림 1.12.4>와 같이 작업1은 A를 B로 변경하고, 작업2는 C를 D로 변경하고, 작업3는 E를 F로 변경한다고 가정해 보자. 당연히 작업1, 작업2, 작업3은 하나의 트랜잭션이다. 여기서 작업3가 실패한 경우에는 작업2의 D를 C로, 작업1의 B를 A로 복구해야 한다. 만약 복구 할 수 없다면 데이터의 비일관성이 발생한다.



<그림 1.12.4>

일관성은 트랜잭션 수행이 끝난 뒤에 트랜잭션에 포함된 작업들의 작업 결과에 대한 데이터 무결성을 말한다. 다음의 <그림 1.12.5>를 보자.

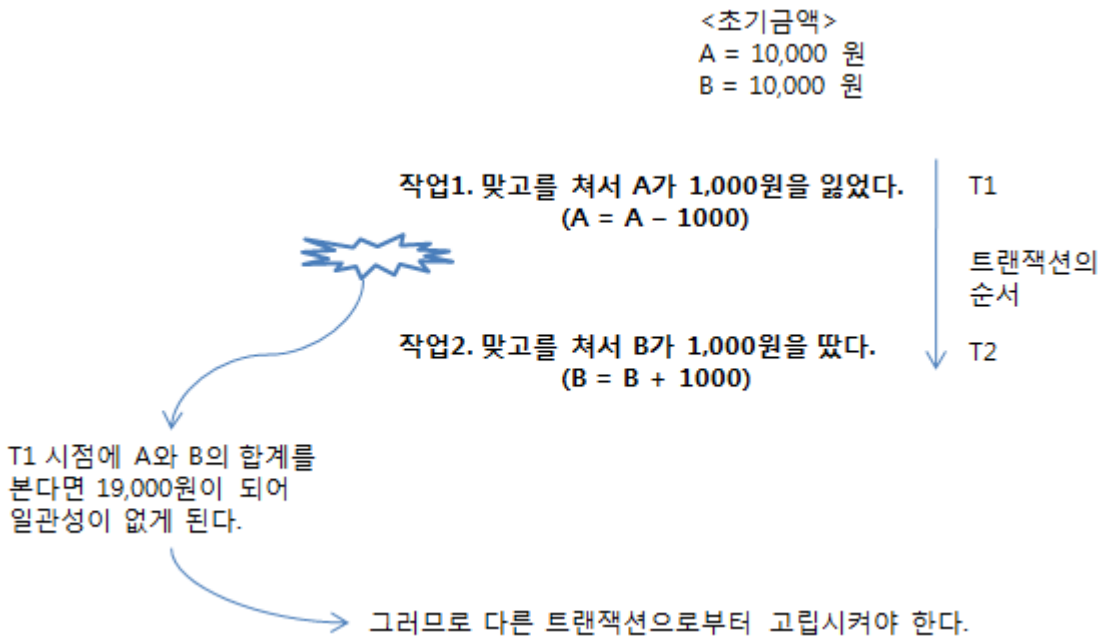


<그림 1.12.5>

2012년 퇴사하지 않은 모든 사원들의 연봉을 10%를 인상한다고 가정하면, [근로계약이력] 테이블

에는 전년도 연봉에서 10%를 인상한 총 4개의 Row가 삽입되어야 한다. 또한 [년도별 연봉] 테이블에는 삽입된 4개의 Row에서 연봉을 총합하여 입력해야 한다. [근로계약이력] 테이블과 [년도별 연봉]에 데이터를 삽입하는 것이 하나의 트랜잭션이라고 가정할 때 트랜잭션이 종료되어도 [근로계약이력] 2012년 연봉의 합은 5830으로 일관되게 유지되어야 한다. 물론 동시에 수행되는 트랜잭션이 없는 상태임을 가정한 것이다. 트랜잭션의 일관성 보장은 사용자나 개발된 어플리케이션의 책임이므로 트랜잭션 제어에 주의를 기울여야 한다.

고립성은 트랜잭션이 동시에 수행될 경우의 충돌 문제를 해결하기 위한 트랜잭션 실행 순서에 관련된 속성이다. 이 속성은 동시성 제어와 관련이 있다. <그림 1.12.6>을 보자.



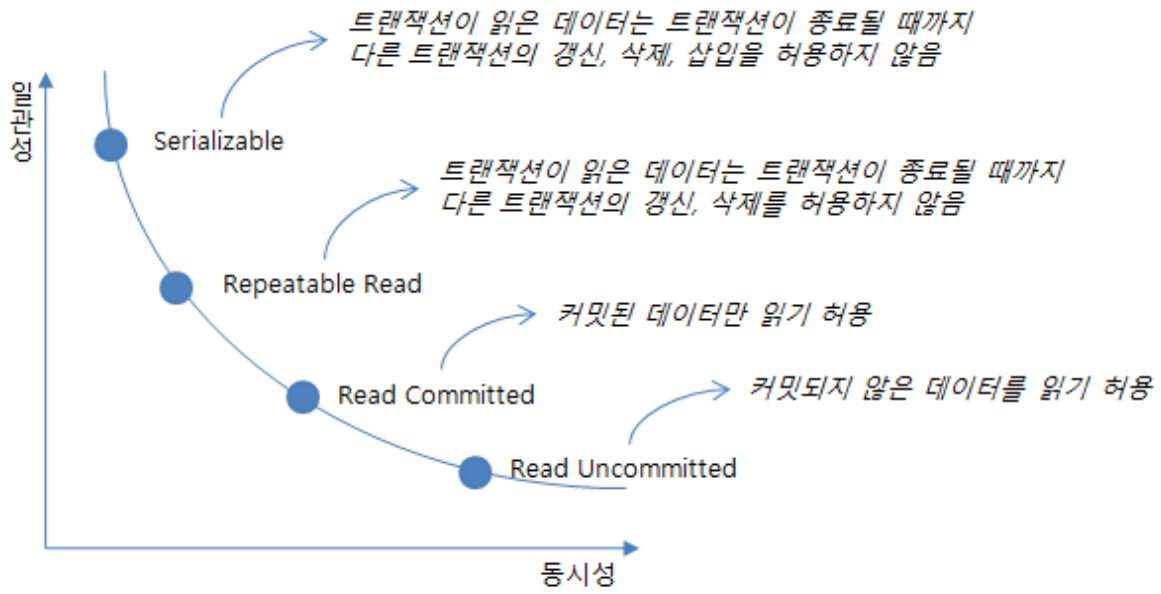
<그림 1.12.6>

<그림 1.12.6>에서 트랜잭션은 작업1, 작업2를 모두 성공적으로 마쳐야 트랜잭션이 성공했다고 볼 수 있다. 만약 작업1을 끝낸 상태에서 A, B의 총 금액을 조회한다면 19,000원이 되어 일관성이 없게 된다. 그러므로 트랜잭션을 고립시켜 트랜잭션이 진행 중일 때는 다른 트랜잭션의 접근을 막아야 한다.

트랜잭션을 얼마만큼 고립시키는지에 따라서 동시성과 일관성도 영향을 받게 된다. 고립화의 수준은 다음의 일관성을 저해하는 3가지 현상에 대한 허용 정도다.

- Dirty Read - 커밋되지 않은 데이터 읽기
- Non-Repeatable Read - 다른 트랜잭션이 갱신, 삭제함으로 인해 데이터를 읽는 시점에 따라 결과가 달라짐
- Phantom Read - 다른 트랜잭션이 삽입함으로 인해 데이터를 읽는 시점에 따라 결과가 달라짐

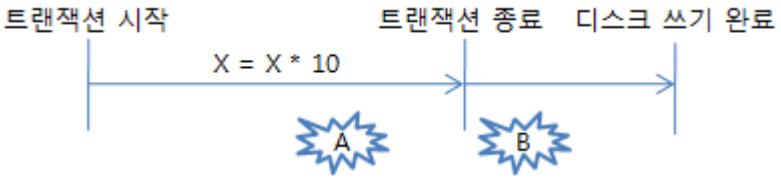
위의 3가지 현상을 얼마나 허용하느냐에 따라 ANSI/ISO에서는 다음의 4가지 고립화 수준을 표준으로 정하였다.



		비일관성 발생 현상		
		Dirty Read	Non-Reatable Read	Phantom Read
고립화수준	Read Uncommitted	O	O	O
	Read Committed	X	O	O
	Repeatable Read	X	X	O
	Serializable	X	X	X

<그림 1.12.7>

영속성은 트랜잭션이 완료된 경우 시스템의 오류가 있더라도 데이터의 손실 없이 데이터베이스에 유지되어야 함을 말한다. 영속성은 회복 시스템이 관리한다. <그림 1.12.7>을 보자.



<그림 1.12.8>

만약 A시점에 시스템에 오류가 발생하여 서비스를 재시작 한 경우에는 시스템은 X * 10 상태가 아닌 X인 상태로 다시 복구해야 한다. 하지만 트랜잭션 완료 시점인 B시점에 시스템이 재시작되었다면 디스크에 쓰기 완료 시점이므로 트랜잭션 로그에 있는 X * 10 이라는 정보를 이용하여 트랜잭션을 재실행하여 X * 10의 값을 유지할 것이다.

동시성 제어 모델

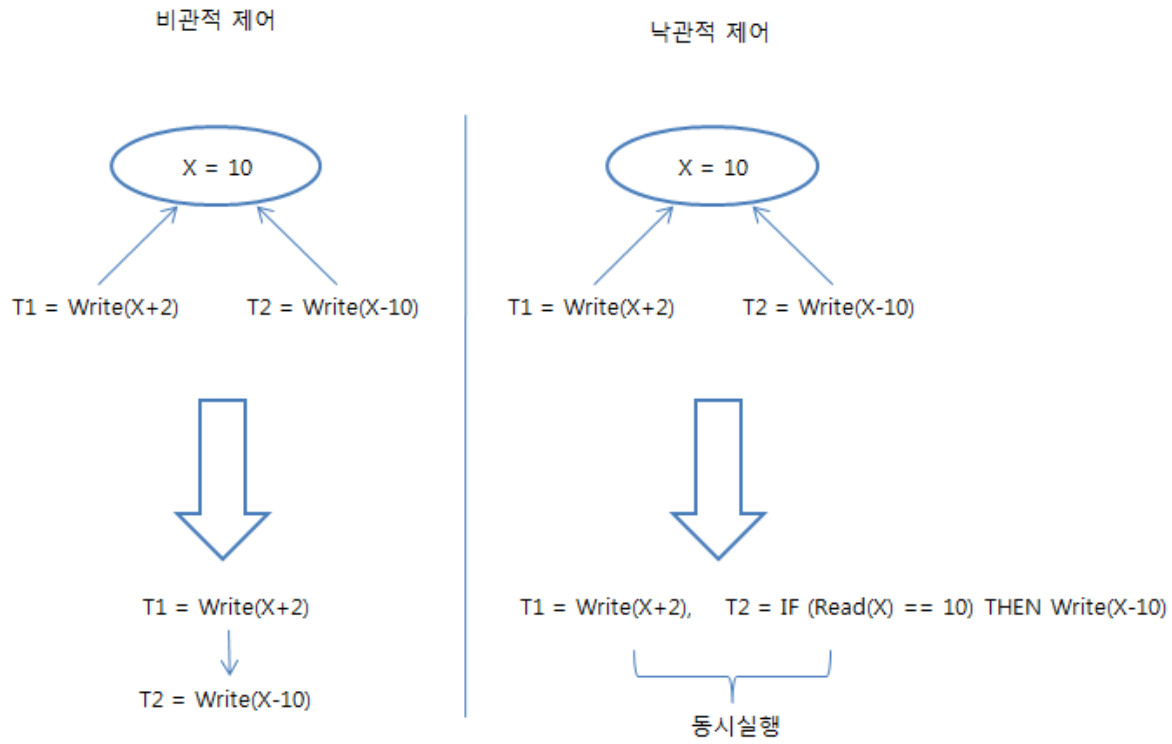
동시성은 한 순간에 여러 개의 트랜잭션이 하나의 데이터를 읽거나 쓰는 작업을 하는 것을 말한다. 앞서 설명한 직렬성은 트랜잭션의 일관성과 관련된 것이다. 일관성은 동시(병렬)에 실행되는 트랜잭션을 직렬화함으로써 보장할 수 있는데, 이렇게 직렬화를 하다 보면 성능 문제가 발생할 수 있다. 그러므로 동시성과 일관성은 트레이드 오프(Trade-Off) 관계다.

동시성을 제어한다는 것은 동시에 실행되는 트랜잭션의 수를 최대하면서 데이터의 일관성을 유지시키는 것으로 다음의 2가지 모델이 있다.

- 비관적 모델 (pessimistic model)
- 낙관적 모델 (optimistic model)

비관적 모델은 두 트랜잭션 T1, T2가 데이터X를 동시에 수정할 수 있음을 가정하는 모델이다. 그러므로 일관성 보장을 위하여 락(lock, 잠금)을 사용한다. 여기서 락이란, 데이터를 수정하는 동안 다른 트랜잭션이 수정하지 못하도록 막는 것을 말한다. 즉, 데이터 X에 락을 사용한다는 것은 T1이 작업을 수행할 때는 다른 트랜잭션이 X를 읽지 못하도록 하는 방법이다. 그러므로 T2는 T1이 락을 해제할 때까지 기다려야지만 X의 값을 읽을 수 있게 된다.

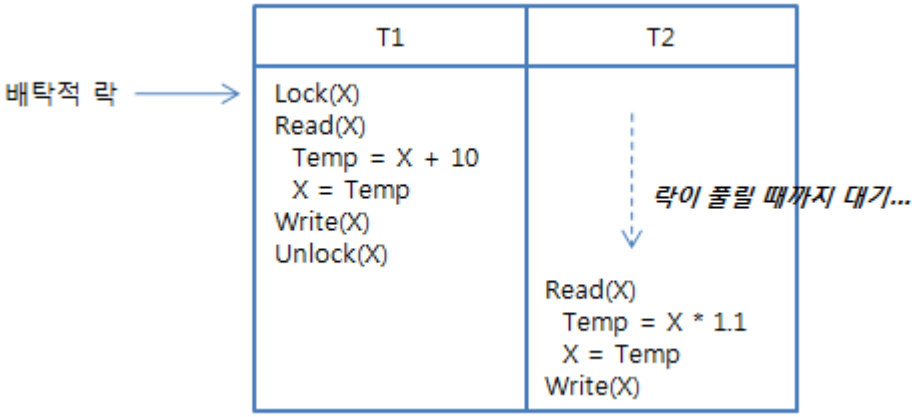
낙관적 모델은 두 트랜잭션 T1, T2가 데이터X를 동시에 수정할 수 없음을 가정하는 모델이다. 그러므로 T1이 작업을 수행할 때도 T2가 X의 값을 읽을 수 있다. 이 모델에서는 읽는 연산이 자유로워져 비관적 모델보다는 동시성이 좋아진다. 하지만, T1이 X를 변경했을 수도 있으므로 만약 T2는 작업을 종료하기 전에 X가 변경되었는지 검사해야 한다.



<그림 1.12.13>

락(Lock)과 MVCC

락(lock)이란, 데이터에 대한 접근을 제어하는 메커니즘이다. 일반적으로 비관적 모델에서 동시성 제어를 위하여 공유 락(Shared Lock)을 사용하는데, 공유 락이란 데이터를 읽을 수는 있지만 갱신이 불가능하게 만드는 것을 말한다. 배타적 락(Exclusive Lock)은 갱신이 될 때까지 일기 접근까지 막는 것을 말한다.

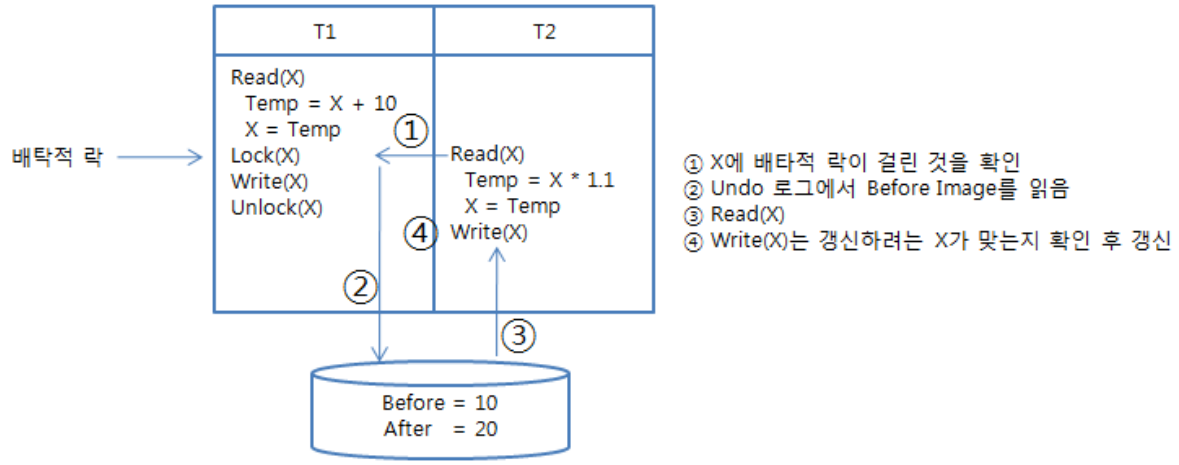


<그림 1.12.14>

공유 락을 사용하면 Dirty Read는 방지할 수 있지만, Non-Repeatable Read난 Phantom Read는 방지할 수 없다. 공유 락은 다른 공유 락과 호환성을 가지지만 배타적 락과는 호환성을 가지지 않으므로 배타적 락을 걸기 위해서는 공유 락들이 모두 해제될 때까지 기다려야 한다.

MVCC(Multi-Version Concurrency Control)은 Dirty Read를 방지하고, 공유 락을 사용하지 않기 때문에 대기 시간이 없어 높은 동시성을 유지할 수 있다. MVCC의 'MV(Multi-Version)'에서 예상 할 수 있듯이 데이터에 대해서 여러 버전을 관리함으로써 트랜잭션의 시작 시간에 따른 데이터의 버전을 읽을 수 있게 한다. 버전 데이터 관리는 DBMS마다 어떻게 구현하느냐에 따라 다르다. 로그 파일로 할 수도 있고, 임시 테이블을 이용할 수도 있다.

MVCC는 높은 동시성을 유지할 수 있지만, 문장 수준의 동시성만 보장하고 트랜잭션 수준의 동시성은 보장하지 않는다. 또한 기본 고립수준(Read Committed)에서도 Non-Repeatable Read, Phantom Read를 방지할 수 있다.



<그림 1.12.15>

Microsoft SQL Server 2005이상 버전이나 Oracle과 같은 DBMS들은 낙관적 모델을 지원한다. 하지만 Microsoft SQL Server 2005는 비관적 모델을 기본으로 하고 있고, Oracle은 낙관적 모델을 기본으로 하고 있다.

참고:

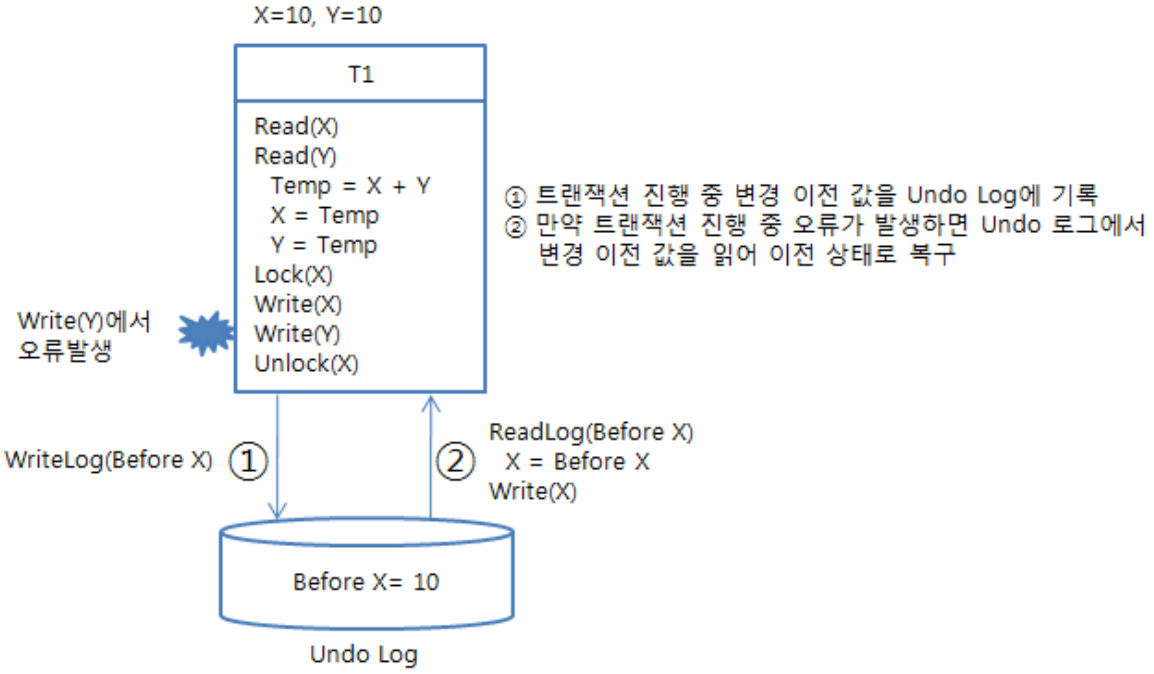
Microsoft SQL Server 2005 이상의 버전에는 2가지 MVCC 방식이 있다. 하나는 트랜잭션 시작시의 행버전을 기준으로 하는 방식이고, 또 하나는 문장이 실행될 때의 행버전을 기준으로 하는 방식이다.

Undo Log와 Redo Log

'Undo' 는 'Un' + 'Do'의 합성어이고, 'Redo'는 'Re' + 'Do'의 합성어다. 'Un'은 '부정'을 뜻하는 접두어다. 'Re'는 '다시'를 뜻하는 접두어다. 'Do'는 실행을 의미하는데, 실행의 대상은 트랜잭션이다. 정리하자면 Undo와 Redo는 다음과 같은 의미가 된다.

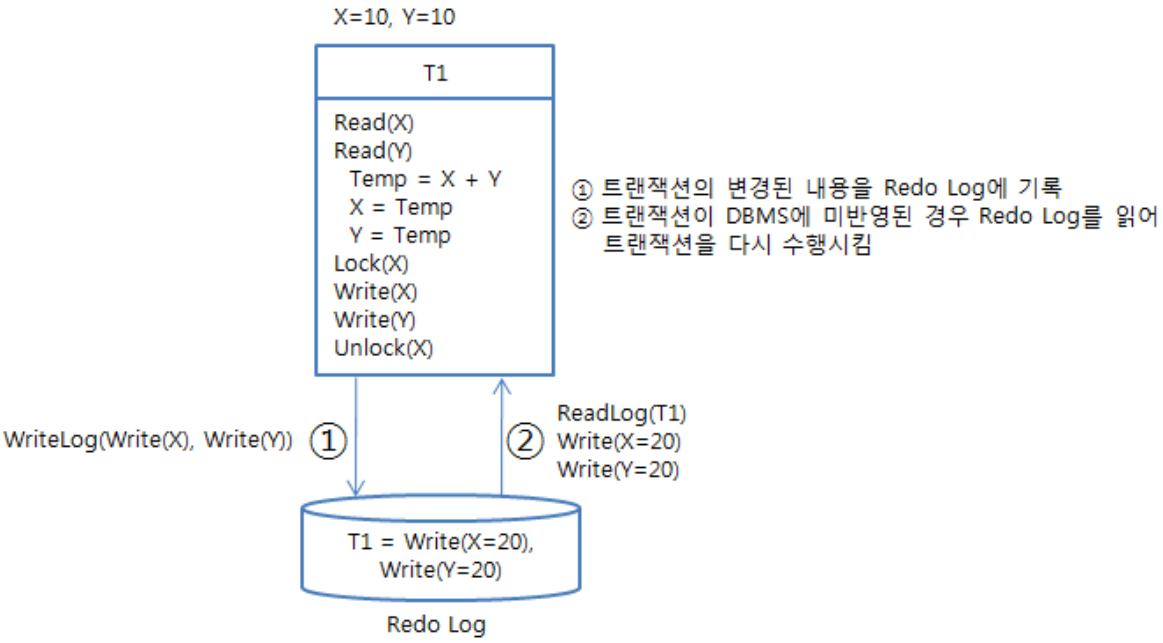
- Undo - 트랜잭션을 실행하지 않음. 즉, 이전 상태로 되돌림
- Redo - 트랜잭션을 다시 실행함. 즉, 원래 변경하려던 대로 변경함.

Undo Log는 다음의 <그림 1.12.16>과 같이 인위적 또는 예기치 못하게 트랜잭션이 커밋(commit)되지 못한 경우, 다시 이전 상태로 되돌리기 위한 기록이다.



<그림 1.12.16>

Redo Log는 다음의 <그림 1.12.17>과 같이 트랜잭션이 커밋 되었지만 예기치 못하게 트랜잭션의 결과를 시스템에 완전히 반영하지 못한 경우, 트랜잭션을 다시 실행시켜 트랜잭션의 결과가 완전하도록 하기 위한 기록이다.



<그림 1.12.17>

위에서 살펴 본 것처럼 DBMS의 복구 시스템은 트랜잭션의 ACID속성을 유지하기 위해 Redo Log와 Undo Log를 남긴다. 일반적으로DBMS의 데이터베이스를 구성하는 파일은 데이터 파일과 로그 파일로 나뉘는데, 복구를 위하여 데이터의 변경사항에 대해서 로그 파일에 먼저 기록을 한다. 그

런 후 주기적으로 데이터 파일에 영구적으로 적용한다. 만약 예기치 못한 일로 DBMS가 셧다운 된다면 DBMS가 서비스를 재시작 할 때Redo Log를 읽어 커밋된 트랜잭션은 다시 트랜잭션을 수행시키고, Undo Log를 읽어 커밋되지 않은 트랜잭션은 이전 상태로 되돌리는 작업을 수행하여 트랜잭션의 ACID 속성을 유지한다.

1.13 IT(Information Technology)

IT는 Information(정보) Technology(기술)의 약자다. 그러므로 IT가 무엇인지 알기 위해서는 우선 Information이 무엇인지 알아야 하고, Technology가 무엇인지 알아야 한다. Information의원천은 Data이므로 Information이 무엇인지 알기 위해서는 Data가 무엇인지 알아야 한다. Data를 시작으로 데이터베이스 범위에서 IT가 무엇인지를 알아보자.

Data, Information

2011년 01월 27일 시점에서 아래의 국립국어원 표준국어대사전의 검색 결과에 따르면 'Data'는' 데이터'라고 표기되며 뜻은 아래와 같다.

데이터(data)

- 이론을 세우는데 기초가 되는 사실. 또는 바탕이 되는 자료.
- 관찰이나 실험, 조사로 얻는 사실이나 정보. '자료 03'으로 순화.
- [컴퓨터] 컴퓨터가 처리할 수 있는 문자, 숫자, 소리, 그림 따위의 형태로 된 정보

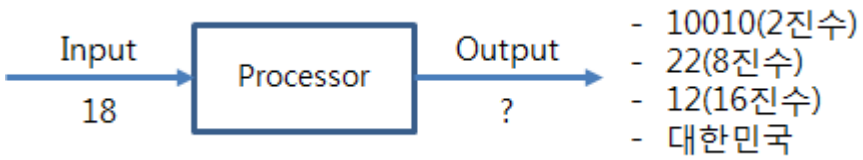
이 중 세 번째를 주목하자. '컴퓨터가 처리할 수 있는 문자, 숫자, 소리, 그림 따위의 형태로 된 정보'라고 되어 있다. 즉, 사전에 따르면 데이터는 정보다. 표기야 동의 할 수 있지만 그 뜻은 동의할 수 없다. 왜냐하면 데이터와 정보는 확연한 차이가 있기 때문이다.

'데이터'와 '정보'의 가장 큰 차이점은 데이터를 처리했느냐, 처리하지 않았느냐의 차이다. 여기서 말하는 '처리'란 정해진 순서와 규칙에 따라 데이터를 추가하거나 재배열, 요약, 삭제하는 행위를 말한다. 용도의 관점에서 보면 의사결정의 근거로 사용되는지의 여부에 따라서도 '데이터'인지 '정보'인지 구분할 수 있다. 데이터를 입력 받은 시스템은 어떤 처리 과정을 거친 결과물로 정보를 출력해 낸다. 그림으로 도식해 보면 <그림 1.13..1>과 같을 것이다.

<그림 1.13..1>

시스템에서는 데이터에 무엇인가를 부가(가치를 더함)하여 '정보'를 생산해 냈다. 시스템이 부가한 것은 정보 소비자가 시스템에 기대한 데이터에 대한 어떤 처리다. <그림 1.13..2>와 같이 10진수

를 2진수 또는 8진수, 16진수로 변환하는 시스템이 있다고 가정 하자.



<그림 1.13..2>

정보 소비자는 Input으로 숫자 '18'을 입력했을 때 2진수로 변환된 값을 Output으로 기대했다고 가정했다면, 시스템에 처리한 결과는 '10010'이 되어야 한다. 하지만 정보 소비자가 시스템으로부터 붙은 결과물이 '22(8진수)'나 문자열 '대한민국' 이라면, 이는 사용자의 기대에 미치지 못하는 결과물이므로 버려지게 될 것이다.

통신 분야에서의 정보이론(information theory)에서는 정보는 조금 다르게 정의되는데, 통신에서의 정보는 그 양으로 판단한다. 정보의 구체적인 의미에 대한 내용은 없다. 수학적으로 이야기하면 정보는 확률로 표현된다. 발생 확률이 적으면 정보량의 많은 것이고, 발생 확률이 1에 가까울수록 정보량은 적다. 예를 들어, 사장님이 다음의 두 가지를 이야기 했을 때 어떤 것이 정보의 양이 많을까 생각해보자.

- 내일은 해가 동쪽에서 뜰 것이다.
- 내일은 사원 모두에게 10%의 인센티브를 지급할 것이다.

첫 번째 문장은 확률이 1에 가까우므로 '정보의 양이 적다(엔트로피 낮음)'라고 말한다. 두 번째는 두 가지 경우를 생각할 수 있다. 만약 이 사실을 몰랐다면 '정보의 양은 매우 많다(엔트로피 높음)'라고 할 수 있다. 만약 미리 알고 있었다면 정보의 양은 몰랐을 때보다는 적다고 이야기 할 수 있을 것이다. 하지만 해가 동쪽에서 뜬다는 것보다는 정보의 양이 훨씬 더 많다는 것은 부정할 수 없다. '내일은 해가 동쪽에서 뜰 것이다'는 것은 모두 다 아는 사실이다. 정확성, 적시성을 충분히 만족시킨다. 하지만 사원들은 '내일은 해가 동쪽에서 뜰 것이다'는 정보에 전혀 관심(관련성)이 없다. 왜냐하면 이는 유질서에 가까움(예측 가능, 발생 확률 높음)을 의미하기 때문이다. 엔트로피가 낮으면 버려지는 정보가 될 확률이 높다.

정보가 의사결정에 이용된다는 관점은 통계적인 관점이다. 일반적으로 비즈니스 부서가 정보를 획득하는 방법은 IT부서에 요청하는 방법 이외에 별다른 방법이 없다. 요즘에는 Self-Service가 가능한 솔루션이 많이 있지만, 기본적으로 데이터를 수집, 통합, 정제하는 일은 IT부서에서 해야만 한다. 즉, 모든 정보 획득에는 비용이 발생한다.

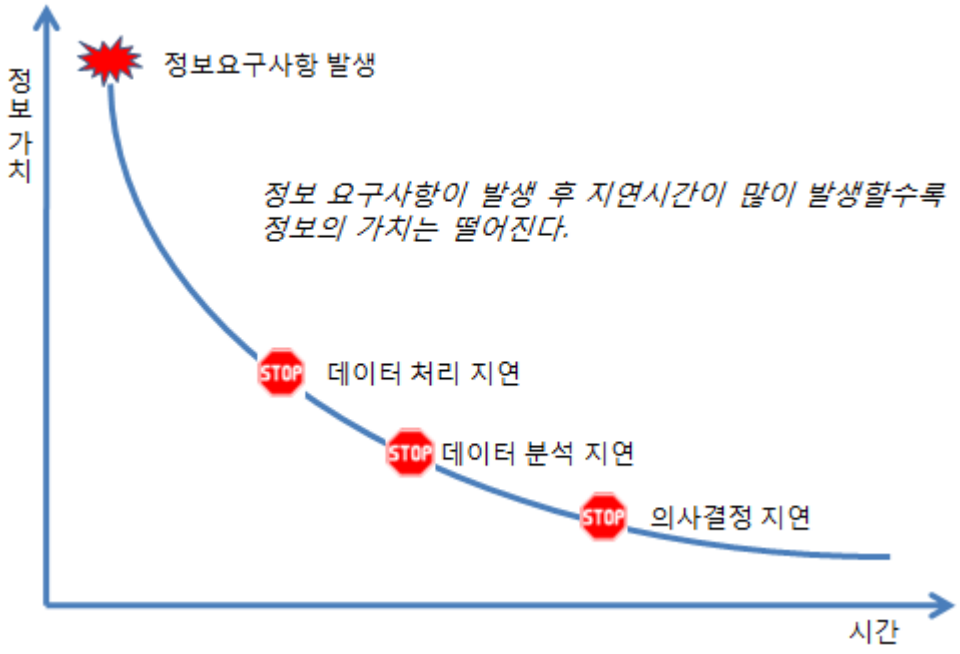
정보의 3요소

앞서 정보 소비자의 기대하는 바와 다른 정보나 엔트로피가 낮은 정보는 쓸모가 없음을 살펴보았다. 쓸모 있는 정보가 되려면 '쓸모가 없음'을 뒤집으면 된다. 정보 소비자의 기대하는 수준의 결

과이거나 엔트로피가 '높다'면 쓸모 있게 된다. 이 때 만족시켜야 하는 3요소가 있는데 이를 '정보의 3요소'라고 한다. 정보의 3요소는 정확성(accuracy), 관련성(relevance), 적시성(timely)을 말하며, 어느 하나라도 만족할 만한 수준에 이르지 못하면 쓸모가 없어질 가능성이 매우 높다.

'정확성'은 처리된 결과가 말 그대로 정확해야 함을 말한다. 예를 들어, 고객ID 'databaser'의 거주지가 '서울'이 실제로도 맞는다면 정확한 값이다. 주민등록번호 7번째 자리가 홀수라면 남자, 짝수라면 여자이어야 한다. 실제로도 주민등록번호 7번째 자리가 '1'일 때 고객의 성별이 실제로도 남자라면 정확한 값이다. <그림 1.3.2>에서는 10진수 '18'을 2진수로 변환해야 하므로 '10010'이 출력되어야만 정확한 값이다.

'적시성'은 정보 소비자가 정보가 필요할 때에 지연 시간이 없어야 함을 의미한다. 요즘에 자주 접하게 되는 RTE(실시간 기업, Real Time Enterprise)에서의 'Real Time'은 '즉시'를 의미하는 것이 아닌 '적시'다. 물론 경우에 따라서 지금 당장 필요할 수도 있다. 이런 경우의 '적시'는 '즉시'다. 지연시간이 많이 발생할수록 정보의 가치는 떨어진다. 아래의 <그림 1.13..3>은 지연시간에 따른 정보의 가치 하락을 설명해주고 있다.



<그림 1.13..3>

물론 이런 요소들 이외에도 기술적인 것이나 결제 및 합의등과 같은 절차에 의한 지연이 발생할 수도 있다. 또한 비즈니스 부서와 IT 부서와의 부적절한 커뮤니케이션등 많은 지연 요소가 있을 수 있다.

'관련성'은 말 그대로 정보 소비자와 관련이 있어야 한다. 예를 들어 검색엔진에서 어떤 검색어를 입력하였으나 전혀 엉뚱한 결과가 나왔다면 관련성이 없어 정보가 될 수 없는 것과 같다. 예전에는 검색엔진에서 '복조(demodulation)'를 검색했을 때 검색 결과는 '복조리'가 거의 대부분이었다. 지금은 검색엔진에서 뱉어내는 결과가 예전에 비하면 아주 좋아졌다.

조금 다른 관점으로 Roman R, Andrus는 다음과 같이 4가지 조건을 만족하면 정보가 가치를 가진다고 했다.

1. 형식 조건: 정보 이용자가 원하는 형태
2. 시간 조건: 위에서 설명한 적시성을 말한다.
3. 공간 or 물리적 접근 조건: 쉽게 정보에 접근할 수 있으면, 쉽게 정보를 전달 할 수 있음
4. 소유 조건: 정보를 소유하여 그 정보를 다른 사람에게 전파하는 것을 통제하면 할수록 가치를 갖는다.

앞서 언급했듯이 정보 획득에는 항상 비용이 발생한다. IT부서는 비즈니스 부서에 요청에 성실히 협조해야 하는 의무를 가진다. 하지만, 이용자의 효용보다 정보 획득 비용이 많이 발생한다면 정보가치는 없고 비용만 낭비하는 꼴이 된다.

기술(Technology)

기술(Technology)이란, 인간이 스스로의 욕구를 만족시키기 위하여 여러 가지 자원의 형태를 변형시킨 것을 기술이라 한다. (중학교 2학년 '기술' 첫 강의 시간에 배우고, 두 번째 시간에 뒤지게 맞아가면서 외운 것이라 잊어버리지도 않는다.) 이 정의를 데이터베이스의 범주에서 다음과 같이 바꾸어 IT를 정의해 보자.

- 욕구 -> 정보욕구 or 요구사항
- 자원 -> 데이터
- 형태를 변형 -> 처리

IT(Information Technology)란, '개인 또는 조직 정보욕구(요구사항)를 만족시키기 위하여 여러 종류(숫자, 문자, 그림 등)의 데이터를 처리하여 정보를 생산하는 것' 쯤으로 정의될 수 있을 것이다.

데이터베이스 시스템이 한 번 움직일 때 마다 많이 일이 벌어진다. 그러므로 데이터베이스 시스템을 이용하여 데이터를 개발하고 관리하는 일은 많은 기술을 필요로 한다. 대장장이가 망치와 불, 쇠를 모두 익숙하게 다루는 것이 필요하듯이 데이터베이스 영역도 종합적인 사고능력에 대한 익숙함이 필요하다. '시스템(system)'에 대해서는 후에 자세히 다룰 예정이다.

정보의 질(Quality)

통신 분야에서는 정보의 양이 많고 적음으로 정보의 질을 좋다 나쁘다를 판단한다. 데이터베이스 범주에서는 정보의 3요소인 정확성, 관련성, 적시성을 모두 만족시켜야만 정보의 질이 좋다고 할 수 있다. 하지만 정보의 질은 측정하기가 매우 어렵다. 혹시나 적시성이 측정하기 가장 쉽다고 할 수 있을지는 모르겠으나 필자의 관점에서는 이것도 어렵다. 왜냐하면 여기서 말하는 적시성은 10초 만에 결과를 받아보았던 것을 0.3초 만에 보게 해달라는 것이 아니기 때문이다. 만약 사업부에서 '내일 이탈이 예상되는 고객들은 누구인가?'를 알고 싶은데 준비가 되지 않았다면 적시성은 이

미 만족시킬 수 없는 것이다. 나에게는 정말 좋은 정보인데 관련성이 없는 다른 사람에게는 그저 처리된 결과물일 뿐 아무짝에도 쓸모 없는 것이 될 수 있는 것이 이기적인 정보의 성질이다. 정확성도 빠질 수 없다. 정확성이 나빠지면 시스템에 대한 신뢰도는 급격하게 떨어지게 되고, 결국엔 더 이상 사용하지 않게 된다.

전문가가 관리하거나 개발한 정보시스템의 정보의 질은 당연히 초급자가 관리하거나 개발한 정보시스템보다 훨씬 좋다. 이 책의 내용들은 모두 정보의 질 향상에 그 목적을 둔다. 정보의 질 향상은 모든 정보시스템의 궁극적인 목표라는 것으로 항상 염두 해 두어야만 수많은 의사결정에서 가치판단을 제대로 할 수 있다.

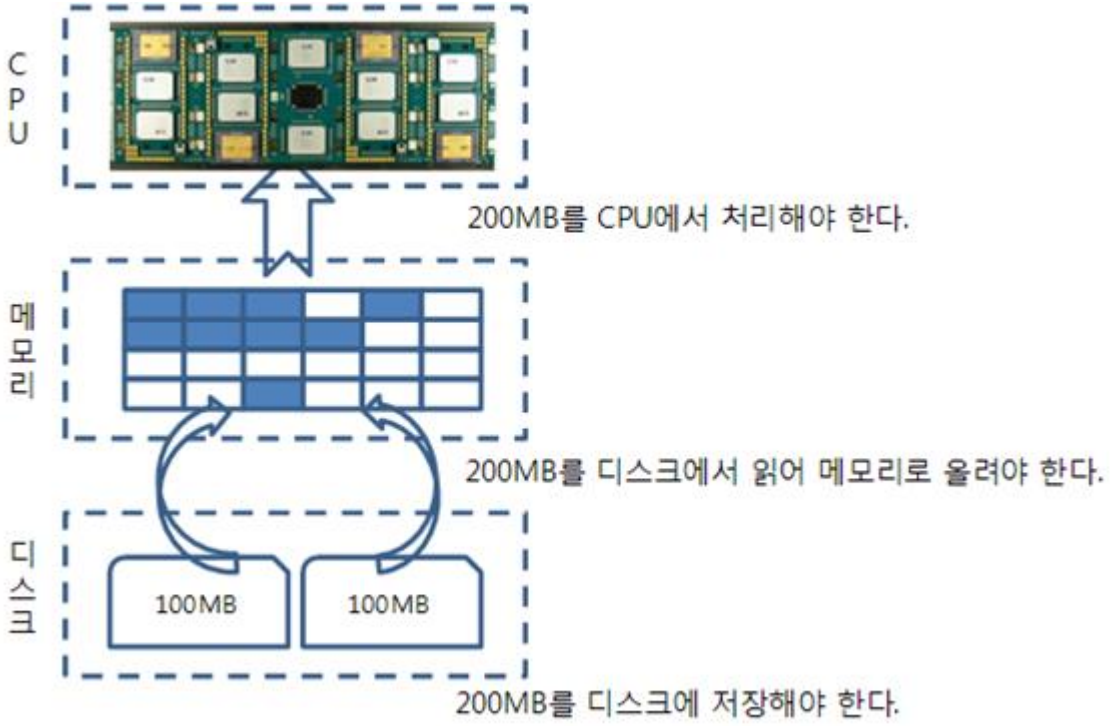
1.14 데이터의 중복과 고립화

데이터 중복으로 인한 성능 저하와 무결성 문제들은 정보시스템의 신뢰성 저하를 가져왔고 이는 데이터베이스의 직접적인 출현배경이 되었다. 데이터 중복의 이유는 정규화되지 않은 데이터 모델의 사용과 시스템간의 고립화로 인한 중복이 대부분이다. 여기에서는 데이터의 중복과 고립화가 어떤 악영향을 끼치는지 알아보도록 하자.

데이터의 중복과 고립화

데이터베이스의 정의에서 '최소한의 중복'이라는 글을 보았을 것이다. 무분별한 데이터의 중복을 없애고 정당화된 데이터의 중복이 필요하다. 여기에서는 데이터의 중복과 고립화가 왜 문제가 되는지 만 정확하게 알면 된다.

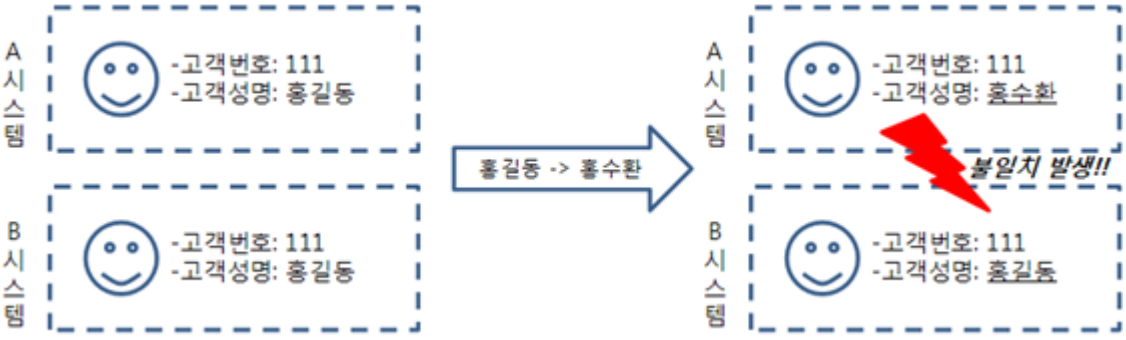
독립적인 정보시스템 내부의 데이터 중복은 필연적으로 성능과 데이터 무결성에 해를 끼치는 현상을 경험하게 해준다. 정보시스템의 최대 성능은 정보시스템을 구성한 하드웨어의 한계를 뛰어넘지 못한다. 최적화라고 해봐야 데이터 처리 알고리즘에 대한 조정뿐이다. 그러므로 데이터의 중복이 성능 저하를 가져오는 것은 명확한 결론을 내릴 수 있게 해준다.



<그림 1.14.1>

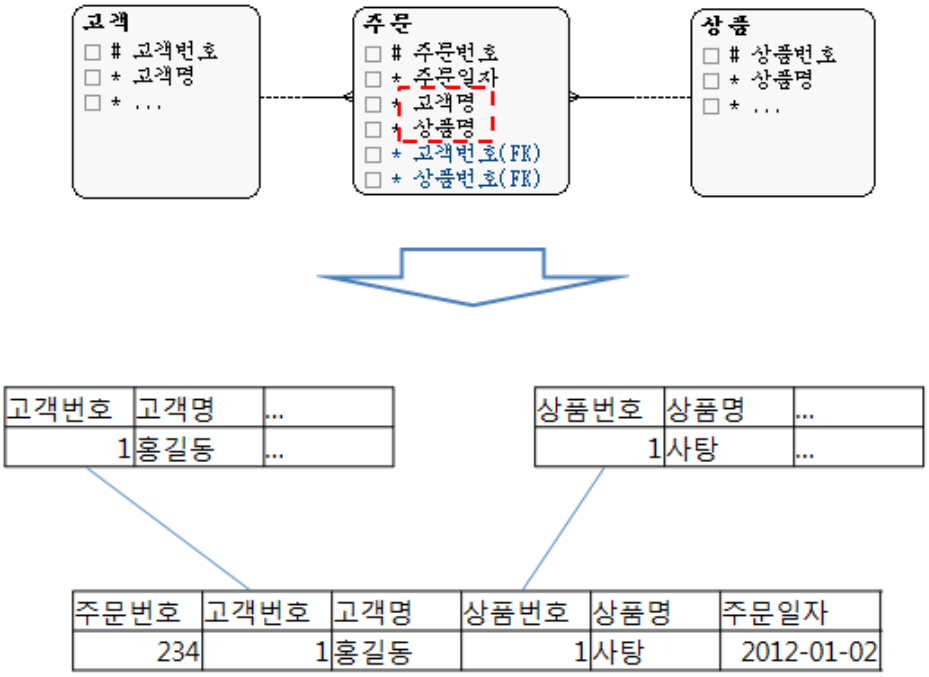
위의 그림과 같이 디스크에 100MB의 데이터가 중복되어 있다고 가정하자. 그러면 100MB 만큼의 저장공간을 낭비해야 한다. 데이터를 처리할 때는 200MB를 읽어야 한다. 즉, 100MB만큼 더 읽어야 한다. 메모리는 100MB 더 필요하고, CPU는 100MB 만큼을 더 처리해야 한다. 만약 메모리가 여유롭지 않다면 공용으로 사용하는 데이터베이스 시스템은 필히 자원의 경합을 불러 일으킨다. 데이터 중복은 비용증가, 성능저하를 반드시 가져오게 된다는 것은 명백하다.

데이터 중복으로 인한 더 큰 문제는 무결성(integrity) 문제다. 무결성 문제는 곧 사용자의시스템에 대한 신뢰를 잃게 됨을 말한다. 사용자가 허용할 수 있는 정도를 벗어나면 결국은 사용자가 없는 죽은 시스템이 된다. 예를 들어 A시스템에서 고객번호 111인 고객의 성명은 '홍길동'이다. B시스템도 마찬가지다. 하지만 시간이 흘러 고객번호 111인 사람이 '홍수환'으로 개명신청을 하였다면 A시스템과 B시스템 모두 '홍수환'으로 갱신되어야 할 것이다. 하지만 데이터의 불일치를 관리하는 장치가 마련되지 않았거나, A시스템과 B시스템이 고립화되었다면 데이터는 불일치 될 것이다. 특히, 이러한 분산 데이터베이스 환경에서는 성능 요소가 중요하다는 이유로 데이터의 중복을 많이 허용한다. 하지만 데이터의 중복에 의한 데이터 품질 저하를 막는 장치는 필요하다.



<그림 1.14.2>

또 다른 예로 물리적으로 다른 데이터베이스 시스템이 아닌 같은 데이터베이스 시스템에서의 데이터 중복도 마찬가지다. 아래의 그림을 보자.



<그림 1.14.3>

고객명, 상품명 이 중복되어 있는 것을 볼 수 있다. 마찬가지로 "고객"의 "홍길동"을 "홍수환"으로 변경하면 "주문"의 "홍길동"도 갱신해 주어야 데이터의 불일치가 발생하지 않는다.

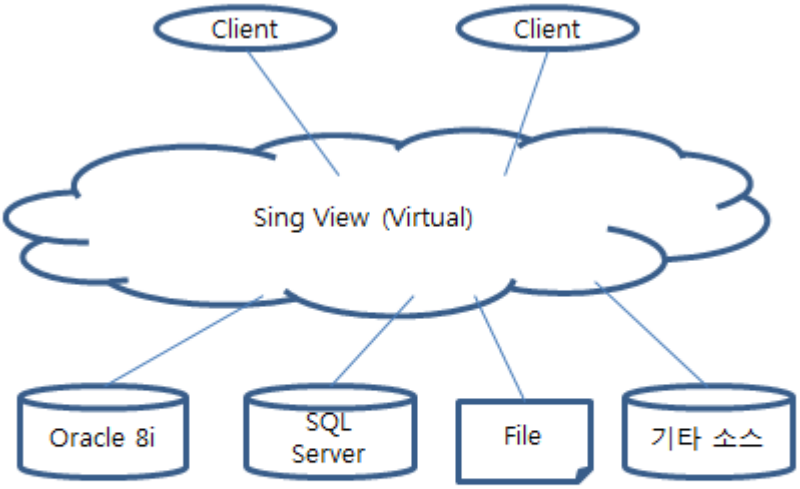
그렇다면 데이터의 중복을 없애는 방법이 없느냐? 있다. 데이터베이스 공부를 해본 사람이라면 다 아는 "정규화"다. 데이터의 중복을 없애는 것은 비용 절감, 성능향상, 무결성 확보라는 3마리 토끼를 잡는 아주 중요한 작업이 된다. 하지만 분산 데이터베이스 환경에서는 쉽지만은 않다. 분산환경에서는 전체/부분중복과 동기/비동기 처리의 문제도 충분히 고려해야 한다. 데이터의 중복을 없애는 일은 우리의 궁극적인 목표인 좋은 정보의 질을 만들어내는 일 중 가장 신경을 많이 써야 하는 어려운 작업이다.

Data Federation, Data Consolidation

데이터의 중복과 고립화를 없애는 일은 데이터베이스 전문가들에게 매우 중요한 일이다. 그러므로 초급자들은 이 부분을 잘 이해해야 하는데, 기술영역이 매우 넓고 전체적인 관점에 서 이해를 해야 하기 때문에 쉽지 않다. 데이터 통합은 기본적으로 데이터의 중복과 고립화를 없애는데, 대표적으로 2가지 방법이 있다.

Data federation은 여러 데이터 저장소에 존재하는 데이터의 물리적 요소를 추상화하여, 단일 액

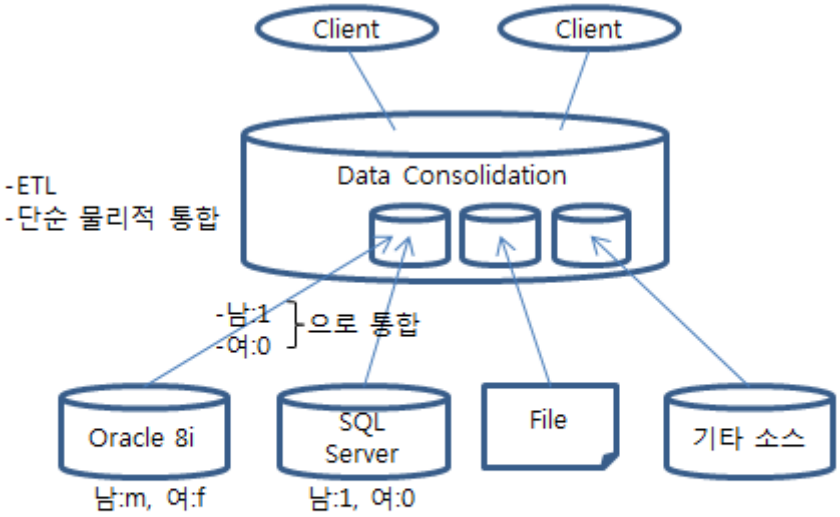
세스 채널을 통해 조회하고 메모리 상에서 통합하는 가상적 통합하는 방식(Gartner)을 말한다. 데이터를 액세스하는 사용자는 가상환경에서 표준화된 단일 뷰(single view)를 제공받음으로써 데이터 통합의 이점(비용감소, 효율성 등)을 얻게 된다.



Data federation(연합)
여러 데이터 저장소에 존재하는 데이터의 물리적 요소를 추상화하여, 단일 액세스 채널을 통해 조회하고 메모리 상에서 통합하는 가상적 통합하는 방식 (Gartner)

<그림 1.14.4>

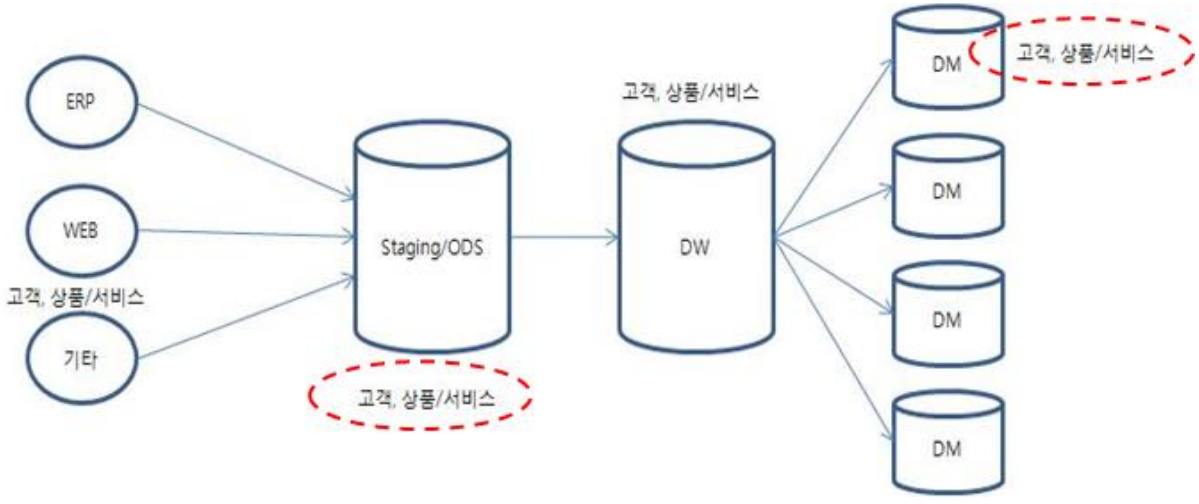
Data Consolidation은 크게 단순한 물리적인 통합과 ETL(Extract, Transform, Load)과정을 거쳐 사용자에게 데이터를 전달하는 2가지 방식이 있다. 당연히 ETL방식이 훨씬 어렵고 복잡하지만, 단순한 물리적인 통합만으로도 매우 큰 효과가 있다. 그 이유는 시너지(시너지는 데이터베이스 시스템에서 다룬다)가 발생하기 때문이다.



<그림 1.14.5>

정보계의 데이터 중복

정보계에서는 데이터의 중복이 필연적으로 일어난다. 데이터의 필연적 중복의 원인은 대량의 Join 문제나 데이터 웨어하우스의 구조적인 특성 때문이다. Join 문제는 대용량 데이터의 정규화로 인해 발생되는데, 이는 데이터의 사용패턴이나 하드웨어의 처리량에 따라서 비정규화 되어 발생하는 문제다. 그러므로 비교적 데이터의 중복은 해결될 수 있다. 하지만 데이터 웨어하우스의 경우 데이터의 흐름이란 것이 있다. 데이터가 운영계로부터 추출되고 최종 사용자에 이르기까지 각 단계별로 데이터의 중복이 발생한다. 아래의 그림은 전통적인 DW의 구조다. (방법론에 따라 구조가 바뀔 수는 있지만 중복은 여전히 존재한다)



<그림 1.14.6>

고객, 상품/서비스가 각 단계별로 중복되어 저장되는 것을 볼 수 있다. 물론 데이터의 형태는 달라질 수 있다. 운영계로부터 데이터가 추출되면 데이터는 각각의 서비스나 업무 관점의 데이터에서 전사적인 관점의 데이터로 통합/변환 된다. 데이터의 형태는 다르지만 본질은 같은 데이터가 중복되는 것이다. 구조적인 데이터의 중복도 물론 피할 수는 있다. 하드웨어의 성능이 비약적으로 발전하였으므로 실제 물리적인 데이터를 1개로 하고, 논리적인 개체(뷰, 스토어드 프로시저 등)로 각각의 서비스나 업무 관점에 맞게 데이터를 제공하면 된다. 물론 ROI를 잘 따져봐야 하는데 그게 쉽지 않다.

1.15 1장 마무리

1 장에서는 전산화, 데이터 모델링 이론과 데이터베이스의 핵심 개념에 대해서 살펴보았다. 독자들은 '전산화'이란 단어가 다소 고리타분하게 들릴 수도 있었을 것이다. 사실 필자도 고리타분하게 들리는데 굳이 '전산화'란 단어를 쓴 이유는 그 개념이 너무나도 중요하기 때문이다. 현실세계의 일부를 컴퓨터세계로 옮겨 놓을 때에 최대한 갭(gap)이 발생하지 않도록 하는 것이 중요하다. 만약 그 갭이 참을 만한 수준이라면 성공적인 것이나, 그렇지 못하다면 실패다. 느려터지고, 나와 별로 관련도 없으며, 결과도 신뢰할 수 없다면 시간과 돈만 낭비한 샘이 된다.