

1. 모델링의 이해

가. 모델링의 정의

인류의 가장 보편적인 특징이면서 욕구 중의 하나는 의사소통을 하면서 항상 그에 대한 기록을 남기는 것이다. 어떤 현상에 대해 기록하고 남겨 자신 스스로 또는 다른 사람에게 적절한 의미를 주기 위해 고대부터 기록의 문화는 발전해 왔다고 할 수 있다. 모델이라고 하는 것은 모형(模型), 축소형(縮小型)의 의미로서 사람이 살아가면서 나타날 수 있는 다양한 현상에 대해서 일정한 표기법에 의해 표현해 놓은 모형이라고 할 수 있다. 이 역시 사람이 어떤 목적을 달성하기 위해 커뮤니케이션의 효율성을 극대화한 고급화된 표현방법으로 설명될 수 있다.

사람이 살아가면서 나타날 수 있는 다양한 현상은 사람, 사물, 개념 등에 의해 발생된다고 할 수 있으며 모델링은 이것을 표기법에 의해 규칙을 가지고 표기하는 것 자체를 의미한다. 즉 모델을 만들어가는 일 자체를 모델링으로 정의할 수 있다.



복잡한 현실세계를 일정한 표기법에 의해 표현하는 일

[그림 1-1-1] 모델링의 정의

다음은 모델링에 대한 다양한 정의를 보여준다.

1) Webster 사전

- 가설적 또는 일정 양식에 맞춘 표현(a hypothetical or stylized representation)
- 어떤 것에 대한 예비표현으로 그로부터 최종 대상이 구축되도록 하는 계획으로서 기여하는 것

2) 복잡한 ‘현실세계’를 단순화시켜 표현하는 것이다. 3) 모델이란 사물 또는 사건에 관한 양상(Aspect)이나 관점(Perspective)을 연관된 사람이나 그룹을 위하여 명확하게 하는 것이다. 4) 모델이란 현실 세계의 추상화된 반영이다.

나. 모델링의 특징

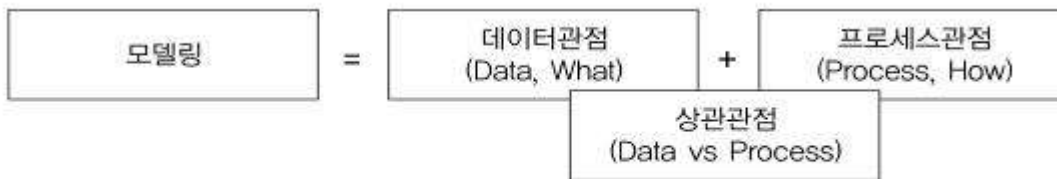
위의 정의를 요약하여 모델링의 특징을 요약하면 추상화, 단순화, 명확화의 3대 특징으로 요약할 수 있다.

1) 추상화(모형화, 가설적)는 현실세계를 일정한 형식에 맞추어 표현을 한다는 의미로 정리할 수 있다. 즉, 다양한 현상을 일정한 양식인 표기법에 의해 표현한다는 것이다. 2) 단순화는 복잡한 현실 세계를 약속된 규약에 의해 제한된 표기법이나 언어로 표현하여 쉽게 이해할 수 있도록 하는 개념을 의미한다. 3) 명확화는 누구나 이해하기 쉽게 하기 위해 대상에 대한 애매모호함을 제거하고 정확(正確)하게 현상을 기술하는 것을 의미한다.

따라서 모델링을 다시 정의하면 ‘현실세계를 추상화, 단순화, 명확화하기 위해 일정한 표기법에 의해 표현하는 기법’으로 정리할 수 있다. 정보시스템 구축에서는 모델링을 계획/분석/설계 할 때 업무를 분석하고 설계하는데 이용하고 이후 구축/운영 단계에서는 변경과 관리의 목적으로 이용하게 된다.

다. 모델링의 세 가지 관점

시스템의 대상이 되는 업무를 분석하여 정보시스템으로 구성하는 과정에서 업무의 내용과 정보시스템의 모습을 적절한 표기법(Notation)으로 표현하는 것을 모델링이라고 한다면, 모델링은 크게 세 가지 관점인 데이터 관점, 프로세스관점, 데이터와 프로세스의 상관관점으로 구분하여 설명할 수 있다.



[그림 1-1-2] 모델링의 관점

1) 데이터 관점 : 업무가 어떤 데이터와 관련이 있는지 또는 데이터간의 관계는 무엇인지에 대해서 모델링하는 방법(What, Data) 2) 프로세스관점: 업무가 실제하고 있는 일은 무엇인지 또는 무엇을 해야 하는지를 모델링하는 방법(How, Process) 3) 데이터와 프로세스의 상관 관점 : 업무가 처리하는 일의 방법에 따라 데이터는 어떻게 영향을 받고 있는지 모델링하는 방법(Interaction)으로 설명될 수 있다.

이 장에서는 데이터 모델링에 대한 기본 개념이 중요하므로 프로세스와 상관모델링에 대한 내용은 생략하고 데이터베이스를 구축하기 위한 데이터 모델링을 중심으로 설명한다.

2. 데이터 모델의 기본 개념의 이해

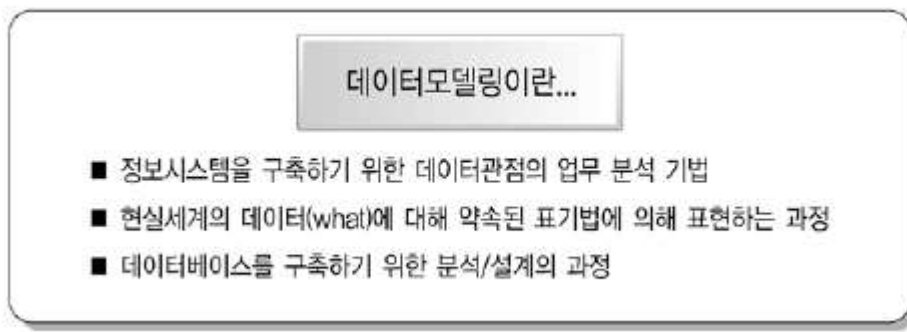
가. 모델링의 정의

데이터 모델은 데이터베이스의 골격을 이해하고 그 이해를 바탕으로 SQL 문장을 기능과 성능적인 측면에서 효율적으로 작성하기 위해 꼭 알아야 하는 핵심요소이다. SQL 전문가를 위한 지식에서도 데이터베이스의 논리적인 구조를 이해하는 데이터 모델을 이해하는 것은 그 다음 SQL 문장을 어떻게 구성할 지에 대한 지식과 효율적인 구성에 대한 밑바탕의 지식을 쌓기 위한 핵심 이론이라 할 수 있다. 일반적으로 데이터 모델링은 다음과 같이 다양하게 정의될 수 있다.

- 정보시스템을 구축하기 위해, 해당 업무에 어떤 데이터가 존재하는지 또는 업무가 필요로 하는 정보는 무엇인지를 분석하는 방법
- 기업 업무에 대한 종합적인 이해를 바탕으로 데이터에 존재하는 업무 규칙(Business Rule)에 대하여 참(True) 또는 거짓(False)을 판별할 수 있는 사실(사실 명제)을 데이터에 접근하는 방법(How), 사람(Who), 전산화와는 별개의(독립적인) 관점에서 이를 명확하게 표현하는 추상화 기법

이것을 좀 더 실무적으로 해석해 보면 업무에서 필요로 하는 데이터를 시스템 구축 방법론에 의해 분석하고 설계하여 정보시스템을 구축하는 과정으로 정의할 수 있다. 데이터 모델링을 하는 주요한 이유는 업무 정보를 구성하는 기초가 되는 정보들을 일정한 표기법에 의해 표현함으로써 정보시스템 구축의 대상이 되는 업무 내용을 정확하게 분석하는 것이 첫 번째 목적이다. 두 번째는 분석된 모델을 가지고 실제 데이터베이스를 생성하여 개발 및 데이터관리에 사용하기 위한 것이다. 즉, 데이터 모델링이라는 것은 단지 데이터베이스만을 구축하기 위한 용도로만 쓰이는 것이 아니라 데

이터 모델링 자체로서 업무를 설명하고 분석하는 부분에도 매우 중요한 의미를 가지고 있다고 할 수 있다.



나. 데이터 모델이 제공하는 기능

업무를 분석하는 관점에서 데이터 모델이 제공하는 기능은 다음과 같다.

- 시스템을 현재 또는 원하는 모습으로 가시화하도록 도와준다.
- 시스템의 구조와 행동을 명세화 할 수 있게 한다.
- 시스템을 구축하는 구조화된 틀을 제공한다.
- 시스템을 구축하는 과정에서 결정한 것을 문서화한다.
- 다양한 영역에 집중하기 위해 다른 영역의 세부 사항은 숨기는 다양한 관점을 제공한다.
- 특정 목표에 따라 구체화된 상세 수준의 표현방법을 제공한다.

3. 데이터 모델링의 중요성 및 유의점

데이터 모델링이 중요한 이유는 파급효과(Leverage), 복잡한 정보 요구사항의 간결한 표현 (Conciseness), 데이터 품질(Data Quality)로 정리할 수 있다.

가. 파급효과(Leverage)

시스템 구축이 완성되어 가는 시점에서 많은 애플리케이션들이 테스트를 수행하고 대규모의 데이터 이행을 성공적으로 수행하기 위한 많은 단위 테스트들이 수행되고 이러한 과정들이 반복된다. 각 단위 테스트들이 성공적으로 수행되고 완료되면 이를 전체를 묶어서 병행 테스트, 통합테스트를 수행하게 된다.

만약, 이러한 시점에 데이터 모델의 변경이 불가피한 상황이 발생한다고 가정해 보자. 이를 위해서 데이터 구조의 변경에 따른 표준 영향 분석, 응용 변경 영향 분석 등 많은 영향 분석이 일어난다. 그 이후에 해당 분야의 실제적인 변경 작업이 발생하게 된다. 변경을 해야 하는 데이터 모델의 형태에 따라서 그 영향 정도는 차이가 있겠지만 이 시기의 데이터 구조의 변경으로 인한 일련의 변경 작업은 전체 시스템 구축 프로젝트에서 큰 위험요소가 아닐 수 없다. 이러한 이유로 인해 시스템 구축 작업 중에서 다른 어떤 설계 과정보다 데이터 설계가 더 중요하다고 볼 수 있다.

복잡한 정보 요구사항의 간결한 표현(Conciseness)

데이터 모델은 구축할 시스템의 정보 요구사항과 한계를 가장 명확하고 간결하게 표현할 수 있는 도구이다. 정보 요구사항을 파악하는 가장 좋은 방법은 수많은 페이지의 기능적인 요구사항을 파악하는 것보다 간결하게 그려져 있는 데이터 모델을 리뷰하면서 파악하는 것이 훨씬 빠른 방법이다. 데이터 모델은 건축물로 비유하자면 설계 도면에 해당한다.

이것은 건축물의 설계 도면이 건축물을 짓는 많은 사람들이 공유하면서 설계자의 생각대로 일사불란하게 움직여서 아름다운 건축물을 만들어 내는 것에 비유할 수 있다. 데이터 모델은 시스템을 구

축하는 많은 관련자들이 설계자의 생각대로 정보요구사항을 이해하고 이를 운용할 수 있는 애플리케이션을 개발하고 데이터 정합성을 유지할 수 있도록 하는 것이다. 이렇게 이상적으로 역할을 할 수 있는 모델이 갖추어야 할 가장 중요한 점은 정보 요구사항이 정확하고 간결하게 표현되어야 한다는 것이다. 우리가 활용하고 있는 데이터 모델이 이와 같은 요소들이 충족된 모델인지를 확인해 볼 필요가 있다.

다. 데이터 품질(Data Quality)

데이터베이스에 담겨 있는 데이터는 기업의 중요한 자산이다. 이 데이터는 기간이 오래되면 될수록 활용가치는 훨씬 높아진다. 그런데 이러한 오래도록 저장된 데이터가 그저 그런 데이터, 정확성이 떨어지는 데이터라고 한다면 어떨까?

이것은 일부 시스템의 기능이 잘못되어 수정하는 성격의 일이 아니다. 이것은 해당 데이터로 얻을 수 있었던 소중한 비즈니스의 기회를 상실할 수도 있는 문제이다. 데이터 품질의 문제가 중요한 이유가 여기에 있다. 데이터 품질의 문제는 데이터 구조가 설계되고 초기에 데이터가 조금 쌓일 때에는 인지하지 못하는 경우가 대부분이다. 이러한 데이터의 문제는 오랜 기간 숙성된 데이터를 전략적으로 활용하려고 하는 시점에 문제가 대두되기 때문이다.

데이터 품질의 문제가 야기되는 중대한 이유 중 하나가 바로 데이터 구조의 문제이다. 중복 데이터의 미정의, 데이터 구조의 비즈니스 정의의 불충분, 동일한 성격의 데이터를 통합하지 않고 분리함으로써의 나타나는 데이터 불일치 등의 데이터 구조의 문제로 인한 데이터 품질의 문제는 치유하기에 불가능한 경우가 대부분이다. 데이터 모델링을 할 때 유의점은 다음과 같다.

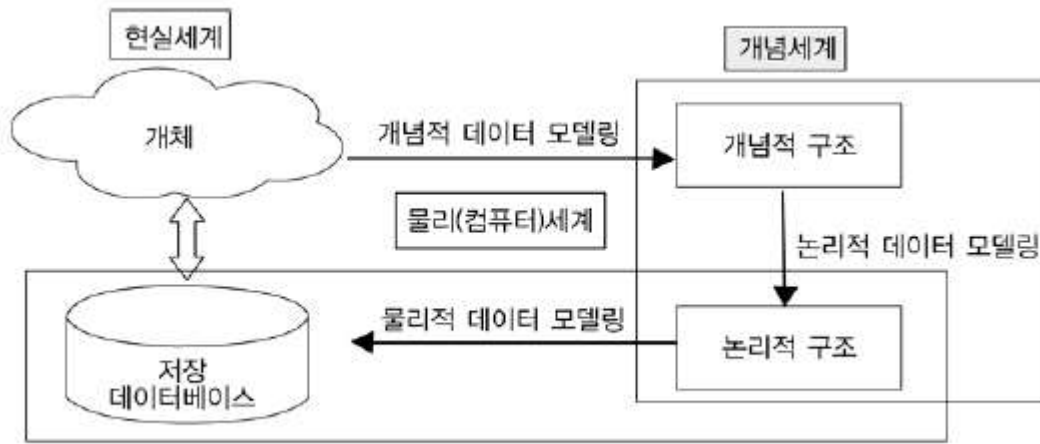
1) 중복(Duplication) 데이터 모델은 같은 데이터를 사용하는 사람, 시간, 그리고 장소를 파악하는데 도움을 준다. 이러한 지식 응용은 데이터베이스가 여러 장소에 같은 정보를 저장하는 잘못을 하지 않도록 한다.

2) 비유연성(Inflexibility) 데이터 모델을 어떻게 설계했느냐에 따라 사소한 업무변화에도 데이터 모델이 수시로 변경됨으로써 유지보수의 어려움을 가중시킬 수 있다. 데이터의 정의를 데이터의 사용 프로세스와 분리함으로써 데이터 모델링은 데이터 혹은 프로세스의 작은 변화가 애플리케이션과 데이터베이스에 중대한 변화를 일으킬 수 있는 가능성을 줄인다.

3) 비일관성(Inconsistency) 데이터의 중복이 없더라도 비일관성은 발생한다. 예를 들어 신용 상태에 대한 갱신 없이 고객의 납부 이력 정보를 갱신하는 것이다. 개발자가 다른 데이터와 모순된다는 고려 없이 일련의 데이터를 수정할 수 있기 때문이다. 데이터 모델링을 할 때 데이터와 데이터 간 상호 연관 관계에 대한 명확한 정의는 이러한 위험을 사전에 예방할 수 있도록 해준다.

4. 데이터 모델링의 3 단계 진행

특히 데이터 모델은 데이터베이스를 만들어내는 설계서로서 분명한 목표를 가지고 있다. 현실세계에서 데이터베이스까지 만들어지는 과정은 [그림 1-1-3]과 같이 시간에 따라 진행되는 과정으로서 추상화 수준에 따라 개념적 데이터 모델, 논리적 데이터 모델, 물리적 데이터 모델로 정리할 수 있다.



[그림 1-1-3] 현실세계와 데이터베이스 사이의 모델

처음 현실세계에서 추상화 수준이 높은 상위 수준을 형상화하기 위해 개념적 데이터 모델링을 전개한다. 개념적 데이터 모델은 추상화 수준이 높고 업무 중심적이고 포괄적인 수준의 모델링을 진행한다.

참고로 EA 기반의 전사적인 데이터 모델링을 전개할 때는 더 상위 수준인 개괄적인 데이터 모델링을 먼저 수행하고 이후에 업무영역에 따른 개념적 데이터 모델링을 전개한다. 엔터티(Entity)중심의 상위 수준의 데이터 모델이 완성되면 업무의 구체적인 모습과 흐름에 따른 구체화된 업무 중심의 데이터 모델을 만들어 내는데 이것을 논리적인 데이터 모델링이라고 한다. 논리적인 데이터 모델링 이후 데이터베이스의 저장 구조에 따른 테이블스페이스 등을 고려한 방식을 물리적인 데이터 모델링이라고 한다. 이것을 요약하여 정리하면 [표 1-1-1]과 같다.

[표 1-1-1] 개념-논리-물리데이터 모델

데이터 모델링	내용	수준
개념적 데이터 모델링	추상화 수준이 높고 업무중심적이고 포괄적인 수준의 모델링 진행. 전사적 데이터 모델링, EA수립시 많이 이용	추상적
논리적 데이터 모델링	시스템으로 구축하고자 하는 업무에 대해 Key, 속성, 관계 등을 정확하게 표현, 재사용성이 높음	
물리적 데이터 모델링	실제로 데이터베이스에 이식할 수 있도록 성능, 저장 등 물리적인 성격을 고려하여 설계	구체적

가. 개념적 데이터 모델링(Conceptual Data Modeling)

개념적 데이터베이스 설계(개념 데이터 모델링)는 조직, 사용자의 데이터 요구사항을 찾고 분석하는 데서 시작한다. 이 과정은 어떠한 자료가 중요하며 또 어떠한 자료가 유지되어야 하는지를 결정하는 것도 포함한다.

이 단계에 있어서의 주요한 활동은 핵심 엔터티와 그들 간의 관계를 발견하고, 그것을 표현하기 위해서 엔터티-관계 다이어그램을 생성하는 것이다. 엔터티-관계 다이어그램은 조직과 다양한 데이터베이스 사용자에게 어떠한 데이터가 중요한지 나타내기 위해서 사용된다. 데이터 모델링 과정이 전 조직에 걸쳐 이루어진다면, 그것은 전사적 데이터 모델(Enterprise Data Model)이라고 불린다. 개념 데이터 모델을 통해 조직의 데이터 요구를 공식화하는 것은 두 가지의 중요한 기능을 지원한다. 첫째, 개념 데이터 모델은 사용자와 시스템 개발자가 데이터 요구 사항을 발견하는 것을 지원한다. 개념 데이터 모델은 추상적이다. 그렇기 때문에 그 모델은 상위의 문제에 대한 구조화를 쉽게 하며, 사용자와 개발자가 시스템 기능에 대해서 논의할 수 있는 기반을 형성한다. 둘째, 개념 데이터 모

델은 현 시스템이 어떻게 변형 되어야 하는가를 이해하는데 유용하다. 일반적으로 매우 간단하게 고립된(Stand Alone) 시스템도 추상적 모델링을 통해서 보다 쉽게 표현되고 설명된다.

나. 논리적 데이터 모델링(Logical Data Modeling)

논리 데이터 모델링은 데이터베이스 설계 프로세스의 Input 으로서 비즈니스 정보의 논리적인 구조와 규칙을 명확하게 표현하는 기법 또는 과정이라 할 수 있다. 논리 데이터 모델링의 결과로 얻어지는 논리 데이터 모델은 데이터 모델링이 최종적으로 완료된 상태라고 정의할 수 있다. 즉 물리적인 스키마 설계를 하기 전 단계의 '데이터 모델' 상태를 일컫는 말이다.

논리 데이터 모델링의 핵심은 어떻게 데이터에 액세스하고 누가 데이터에 액세스하며, 그러한 액세스의 전산화와는 독립적으로 다시 말해서 누가(Who), 어떻게(How: Process) 그리고 전산화와는 별개로 비즈니스 데이터에 존재하는 사실들을 인식하여 기록하는 것이다.

데이터 모델링 과정에서 가장 핵심이 되는 부분이 논리 데이터 모델링이라고 할 수 있다. 데이터 모델링이란 모델링 과정이 아닌 별도의 과정을 통해서 조사하고 결정한 사실을 단지 ERD 라는 그림으로 그려내는 과정을 말하는 것이 아니다. 시스템 구축을 위해서 가장 먼저 시작할 기초적인 업무 조사를 하는 초기단계에서부터 인간이 결정해야 할 대부분의 사항을 모두 정의하는 시스템 설계의 전 과정을 지원하는 '과정의 도구' 라고 해야 할 것이다.

이 단계에서 수행하는 또 한가지 중요한 활동은 정규화이다. 정규화는 논리 데이터 모델 상세화 과정의 대표적인 활동으로, 논리 데이터 모델의 일관성을 확보하고 중복을 제거하여 속성들이 가장 적절한 엔터티에 배치되도록 함으로써 보다 신뢰성 있는 데이터구조를 얻는데 목적이 있다. 논리 데이터 모델의 상세화는 식별자 확정, 정규화, M:M 관계 해소, 참조 무결성 규칙 정의 등을 들 수 있으며, 추가적으로 이력 관리에 대한 전략을 정의하여 이를 논리 데이터 모델에 반영함으로써 데이터 모델링을 완료하게 된다.

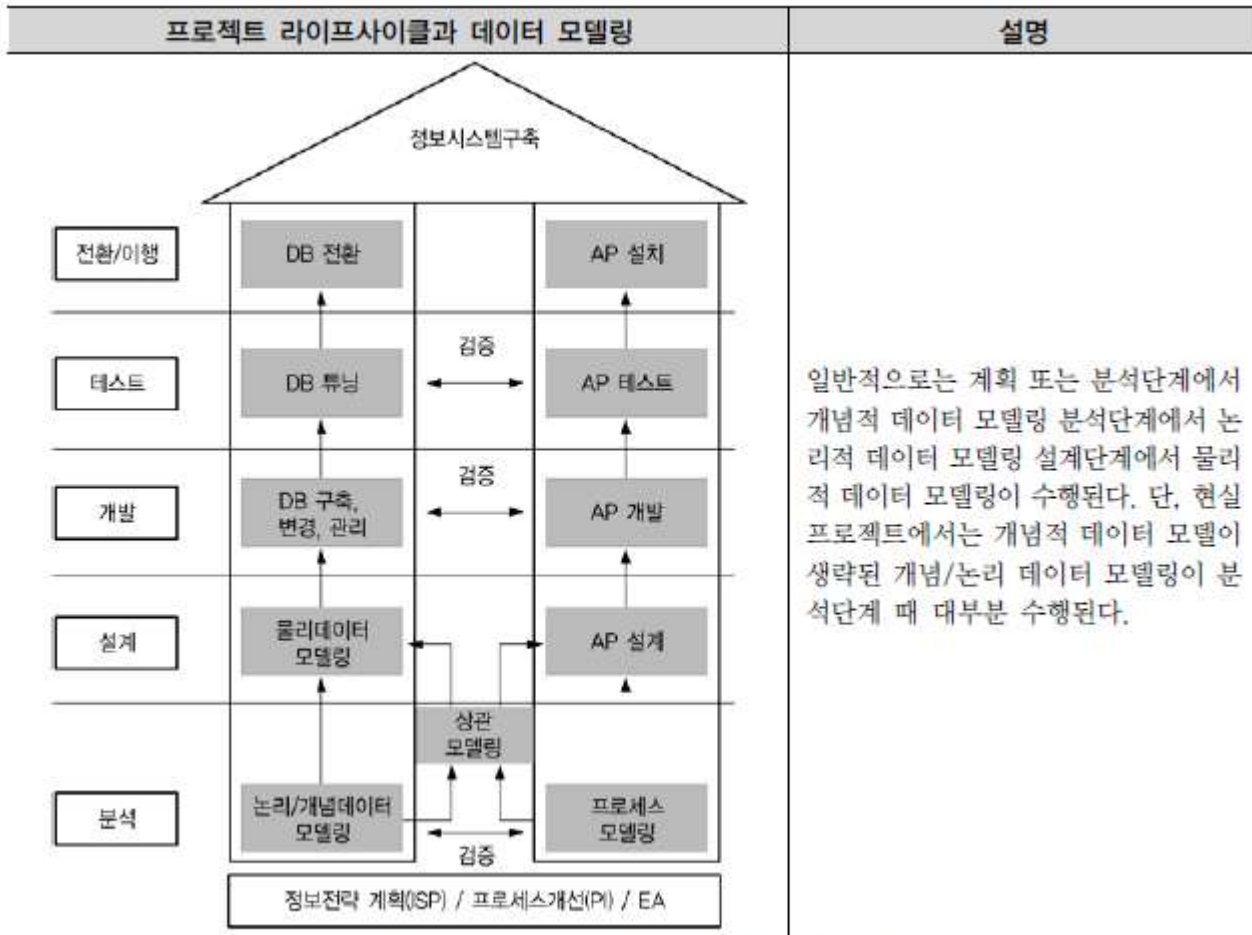
다. 물리적 데이터 모델링(Physical Data Modeling)

데이터베이스 설계 과정의 세 번째 단계인 물리 데이터 모델링은 논리 데이터 모델이 데이터 저장소로서 어떻게 컴퓨터 하드웨어에 표현될 것인가를 다룬다. 데이터가 물리적으로 컴퓨터에 어떻게 저장될 것인가에 대한 정의를 물리적 스키마라고 한다. 이 단계에서 결정되는 것은 테이블, 칼럼 등으로 표현되는 물리적인 저장 구조와 사용될 저장 장치, 자료를 추출하기 위해 사용될 접근 방법 등이 있다. 계층적 데이터베이스 관리 시스템 환경에서는 데이터베이스 관리자가 물리적 스키마를 설계하고 구현하기 위해서 보다 많은 시간을 투자하여야 한다.

실질적인 현실 프로젝트에서는 개념적 데이터 모델링 논리적 데이터 모델링 물리적 데이터 모델링으로 수행하는 경우는 드물며 개념적 데이터 모델링과 논리적 데이터 모델을 한꺼번에 수행하여 논리적인 데이터 모델링으로 수행하는 경우가 대부분이다. 프로젝트 생명 주기에 따른 일반적인 데이터 모델은 다음과 같이 수행된다.

5. 프로젝트 생명주기(Life Cycle)에서 데이터 모델링

Waterfall 기반에서는 데이터 모델링의 위치가 분석과 설계 단계로 구분되어 명확하게 정의할 수 있다. 정보공학이나 구조적 방법론에서는 보통 분석단계에서 업무 중심의 논리적인 데이터 모델링을 수행하고 설계단계에서 하드웨어와 성능을 고려한 물리적인 데이터 모델링을 수행하게 된다. 나선형 모델, 예를 들어 RUP(Rational Unified Process 나 마르미)에서는 업무 크기에 따라 논리적 데이터 모델과 물리적 데이터 모델이 분석, 설계단계 양쪽에서 수행이 되며 비중은 분석단계에서 논리적인 데이터 모델이 더 많이 수행되는 형태가 된다.



[그림 1-1-4] 프로젝트 생명주기에 따른 데이터 모델

데이터 축과 애플리케이션축으로 구분되어 프로젝트가 진행되면서 각각에 도출된 사항은 상호 검증을 지속적으로 수행하면서 단계별 완성도를 높게 된다. 단, 객체지향 개념은 데이터와 프로세스를 한꺼번에 바라보면서 모델링을 전개하므로 데이터 모델링과 프로세스 모델링을 구분하지 않고 일체형으로 진행(대표적인 예가 데이터(속성)과 프로세스(Method)가 같이 있는 클래스(Class))하게 된다.

6. 데이터 모델링에서 데이터독립성의 이해

가. 데이터독립성의 필요성

일체적 구성에서 기능화 된 구성의 가장 큰 목적은 상호간 영향에서 벗어나 개별 형식이 가지는 고유의 기능을 유지시키며 그 기능을 극대화하는 것이다. 컴포넌트 기반의 모듈 구성도 각각이 고유한 기능을 가지면서 다른 기능을 가지고 있는 컴포넌트와 인터페이스를 가지게 하는 모습으로 정의할 수 있다. SOA의 '서비스'라고 하는 단위도 독립적인 비즈니스로 처리 가능한 단위를 서비스로 정의하고 그것이 다른 서비스에 비해 독립성을 구성하여 개별로도 의미가 있고 다른 서비스와 결합하여 프로세스로 제공해도 의미가 있는 단위(예, BPM)로 제공하게 된다.

이와 같이 어떤 단위에 대해 독립적인 의미를 부여하고 그것을 효과적으로 구현하게 되면 자신이 가지는 고유한 특징을 명확하게 할 뿐만 아니라 다른 기능의 변경으로부터 쉽게 변경되지 않고 자신의 고유한 기능을 가지고 기능을 제공하는 장점을 가지게 된다.

데이터독립성을 이해하기 위해서는, 데이터 독립성이라고 하는 개념이 출현하게 된 배경을 이해할 필요가 있다. 데이터독립성의 반대말은 데이터종속성이다. 여기에서 종속의 주체는 보통 응용(Application)을 지칭하는 경우이다. 응용(Application)은 사용자 요구사항을 처리하는 사용자 접

점의 인터페이스 오브젝트이다. 과거에 파일방식으로 데이터를 구성할 때는 데이터가 있는 파일과 데이터에 접근하기 위한 인덱스를 별도로 구현하여 접근하게 하였는데 사용자가 접근하는 방법(트랜잭션의 유형)에 따라 파일의 정렬 순서, 인덱스의 정렬 순서, 파일 구성 등을 제공하기 쉽게 별도로 구성하였다.

즉, 사용자 접근하는 유형에 따라 데이터를 구성하는 방법이 영향을 받게 된다. 메인 프레임 환경에서 파일 방식을 사용하여 데이터처리를 했던 메인 프레임 세대는 개별로 처리했던 접근방법을 이해할 수 있으나 1990년대 이후의 Client/Sever 이후 세대는 파일 처리 방식 이해가 난해할 수도 있다. 데이터독립성은 지속적으로 증가하는 유지보수 비용을 절감하고 데이터 복잡도를 낮추며 중복된 데이터를 줄이기 위한 목적이 있다. 또한 끊임없이 요구되는 사용자 요구사항에 대해 화면과 데이터베이스 간에 서로 독립성을 유지하기 위한 목적으로 데이터 독립성 개념이 출현했다고 할 수 있다.



[그림 1-1-5] 데이터독립성의 필요성

데이터독립성은 미국 표준 협회(ANSI) 산하의 X3 위원회(컴퓨터 및 정보 처리)의 특별연구분과위원회에서 1978년에 DBMS와 그 인터페이스를 위해 제안한 'three-schema architecture'로 정의할 수 있다.

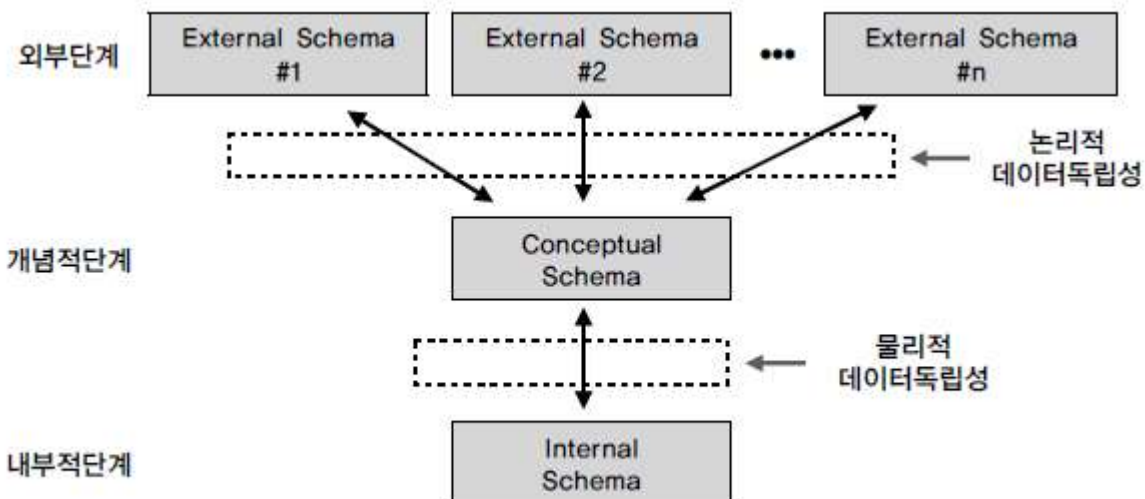
데이터독립성을 확보하게 되면 다음과 같은 효과를 얻을 수 있다.

- 각 View의 독립성을 유지하고 계층별 View에 영향을 주지 않고 변경이 가능하다.
- 단계별 Schema에 따라 데이터 정의어(DDL)와 데이터 조작어(DML)가 다를 수 있다.

데이터독립성을 이해하기 위해서는 3단계로 표현된 ANSI 표준 모델을 살펴보면 되는데 특히 3단계인 구조, 독립성, 사상(Mapping) 3가지를 이해하면 된다.

나. 데이터베이스 3단계 구조

ANSI/SPARC의 3단계 구성의 데이터독립성 모델은 외부 단계와 개념적 단계, 내부적 단계로 구성된 서로 간섭되지 않는 모델을 제시하고 있다.



[그림 1-1-6] 데이터독립성

데이터독립성의 3 단계에서 외부 단계는 사용자와 가까운 단계로 사용자 개개인이 보는 자료에 대한 관점과 관련이 있는 부분이다. 즉, 사용자가 처리하고자 하는 데이터 유형과 관점, 방법에 따라 다른 스키마 구조를 가지고 있는 것이다. 개념 단계는 사용자가 처리하는 데이터 유형의 공통적인 사항을 처리하는 통합된 뷰를 데이터 유형의 공통적인 사항을 처리하는 통합된 뷰를 스키마 구조로 디자인한 형태이다. 우리가 쉽게 이해하는 데이터 모델은 사용자가 처리하는 통합된 뷰를 설계하는 도구라고 이해해도 무방하다. 마지막으로 내부적 단계는 데이터가 물리적으로 저장된 방법에 대한 스키마 구조를 말한다. 다음에서 [그림 1-1-6]의 3 단계 구조의 상세 사항에 대해 각 구성별로 예를 들어 설명한다.

다. 데이터독립성 요소

[표 1-1-2] 데이터독립성 구성요소

항목	내용	비고
외부스키마 (External Schema)	<ul style="list-style-type: none"> - View 단계 여러 개의 사용자 관점으로 구성, 즉 개개 사용자 단계로써 개개 사용자가 보는 개인적 DB 스키마 - DB의 개개 사용자나 응용프로그래머가 접근하는 DB 정의 	사용자 관점 접근하는 특성에 따른 스키마 구성
개념스키마 (Conceptual Schema)	<ul style="list-style-type: none"> - 개념단계 하나의 개념적 스키마로 구성 모든 사용자 관점을 통합한 조직 전체의 DB를 기술하는 것 - 모든 응용시스템들이나 사용자들이 필요로 하는 데이터를 통합한 조직 전체의 DB를 기술한 것으로 DB에 저장되는 데이터와 그들간의 관계를 표현하는 스키마 	통합관점
내부스키마 (Internal Schema)	<ul style="list-style-type: none"> - 내부단계, 내부 스키마로 구성, DB가 물리적으로 저장된 형식 - 물리적 장치에서 데이터가 실제적으로 저장되는 방법을 표현하는 스키마 	물리적 저장구조

데이터베이스 스키마 구조는 3 단계로 구분되고 각각은 상호 독립적인 의미를 가지고 고유한 기능을 가진다. 데이터 모델링은 통합관점의 뷰를 가지고 있는 개념 스키마를 만들어가는 과정으로 이해할 수 있다.

라. 두 영역의 데이터독립성

이렇게 3 단계로 개념이 분리되면서 각각의 영역에 대한 독립성을 지정하는 용어가 바로 논리적인 독립성과 물리적인 독립성이다.

[표 1-1-3] 논리적, 물리적 데이터독립성

독립성	내용	특징
논리적 독립성	<ul style="list-style-type: none"> - 개념 스키마가 변경되어도 외부 스키마에는 영향을 미치지 않도록 지원하는 것 - 논리적 구조가 변경되어도 응용 프로그램에 영향 없음 	<ul style="list-style-type: none"> - 사용자 특성에 맞는 변경가능 - 통합 구조 변경가능
물리적 독립성	<ul style="list-style-type: none"> - 내부스키마가 변경되어도 외부/개념 스키마는 영향을 받지 않도록 지원하는 것 - 저장장치의 구조변경은 응용프로그램과 개념스키마에 영향 없음 	<ul style="list-style-type: none"> - 물리적 구조 영향 없이 개념구조 변경가능 - 개념구조 영향 없이 물리적인 구조 변경가능

즉, 논리적인 데이터독립성은 외부의 변경에도 개념스키마가 변하지 않는 특징을 가진다. 물론, 새로운 요건이 추가되거나 삭제될 경우 칼럼이 변형될 수 있지만 그러한 변화가 개별 화면이나 프로세스에 의해 변화된다기 보다는 전체 업무적인 요건을 고려하여 종합적으로 영향을 받음을 의미한다.

마. 사상(Mapping)

영어로 'Mapping'은 우리말로 '사상'이라고 번역되는데 이것은 상호 독립적인 개념을 연결시켜주는 다리를 뜻한다. 데이터독립성에서는 크게 2가지의 사상이 도출된다.

[표 1-1-4] 사상(Mapping)

사상	내용	예
외부적/개념적 사상 (논리적 사상)	- 외부적 뷰와 개념적 뷰의 상호 관련성을 정의함	사용자가 접근하는 형식에 따라 다른 타입의 필드를 가질 수 있음. 개념적 뷰의 필드 타입은 변화가 없음
개념적/내부적 사상 (물리적 사상)	- 개념적 뷰와 저장된 데이터베이스의 상호관련성 정의	만약 저장된 데이터베이스 구조가 바뀐다면 개념적/내부적 사상이 바뀌어야 함. 그래야 개념적 스키마가 그대로 남아있게 됨

즉, 외부 화면이나 사용자에게 인터페이스하기 위한 스키마 구조는 전체가 통합된 개념적 스키마와 연결된다는 것이 논리적 사상이다. 또한 통합된 개념적 스키마 구조와 물리적으로 저장된 구조의 물리적인 테이블스페이스와 연결되는 구조가 물리적 사상이다. 데이터독립성을 보장하기 위해서는 사상을 하는 스크립트(DDL)를 DBA가 필요할 때마다 변경해 주어야 한다. 즉, 각 단계(외부, 개념적, 내부적)의 독립성을 보장하기 위해서 변경사항이 발생했을 때 DBA가 적절하게 작업을 해주기 때문에 독립성이 보장된다고도 할 수 있다.

7. 데이터 모델링의 중요한 세 가지 개념

가. 데이터 모델링의 세 가지 요소

데이터 모델링을 구성하는 중요한 개념 세 가지가 있는데 이것은 데이터 모델에 대한 이해의 근간이 되므로 반드시 기억할 필요가 있다.

1) 업무가 관여하는 어떤 것(Things) 2) 어떤 것이 가지는 성격(Attributes) 3) 업무가 관여하는 어떤 것 간의 관계(Relationships)

이 세 가지는 데이터 모델링을 완성해 가는 핵심 개념으로서 결국 엔터티, 속성, 관계로 인식되는 것이다. 사물이나 사건 등을 바라 볼 때 전체를 지칭하는 용어를 어떤 것(Things)이라 하고, 그 어떤 것이 가지는 세부적인 사항을 성격(Attributes)이라고 할 수 있다. 또한 각각의 어떤 것은 다른 어떤 것과 연관성을 가질 수 있는데 이것을 관계(Relationship)라고 표현한다.

예를 들어 '이주일과 심순애가 존재하고 둘 사이는 서로 사랑하는 연인사이이다. 이주일은 키가 180cm에 성격은 친절하고 심순애는 키가 165cm에 세심하며 활발한 성격을 가지고 있다'는 시나리오를 살펴보자. 여기에서 '이주일, 심순애'는 어떤 것(Things)에 해당하고 '사랑하는 연인사이'가 어떤 것 간의 관계(Relationships)에 해당하며 '180cm에 성격은 친절, 세심하며 활발함'이 어떤 것이 가지는 성격(Attributes)에 해당한다.

위의 예와 같이 이 세상의 모든 사람, 사물, 개념 등은 어떤 것, 어떤 것 간의 관계, 성격의 구분을 통해서 분류할 수 있다. 바로 이러한 원리, 즉 자연계에 존재하는 모든 유형의 정보들을 세 가지 관점의 접근 방법을 통해 모델링을 진행하는 것이다.

나. 단수와 집합(복수)의 명명

데이터 모델링에서는 이 세 가지 개념에 대해서 단수와 복수의 개념을 분명하게 구분하고 있고 실제로 데이터 모델링을 할 때 많이 활용되는 용어이다.

[표 1-1-5] 용어의 구분정의

개념	복수/집합개념 타입/클래스	개별/단수개념 어커런스/인스턴스
어떤 것 (Thing)	엔터티 타입(Entity Type)	엔터티(Entity)
	엔터티(Entity)	인스턴스(Instance), 어커런스(Occurrence)
어떤 것간의 연관 (Association between Things)	관계(Relationship)	페어링(Pairing)
어떤 것의 성격 (Characteristic of a Thing)	속성(Attribute)	속성값(Attribute Value)

어떤 것의 전체를 지칭하는 것을 영문으로 Entity Set, Entity Type 의 복수의 의미를 가지는 Set 이나 Type 을 포함하여 표현하기도 한다. 그래서 엔터티 타입으로 표현하기도 하는데 실제 실무 현장에서는 엔터티로 짧게 명명한다.

즉 엔터티를 어떤 것에 대한 집합을 명명하여 지칭한다. 어떤 것에 대한 개별 지칭으로 엔터티가 단수명사로서 단어의 의미가 있지만, 엔터티를 집합개념으로 사용하는 경우에는 개별요소에 대해서는 인스턴스/어커런스를 단수의 개념으로 사용하여 부른다.

관계(Relationship)도 이를 복수로 통칭하여 관계로 표현하고 관계에 포함된 개별 연관성을 페어링 이라고 부르기도 한다. 그러나 페어링이라는 용어는 실제 데이터 모델링을 할 때는 잘 사용하지 않으며 그냥 일반적으로 단수든 복수든 관계라고 표현하는 경우가 많다. 어떤 것이 가지는 성격 (Attribute)에 대한 집합개념이 속성이고 그 안에 개별 값들을 속성값으로 구분하여 복수와 단수의 개념으로 구분할 수 있다. 본 가이드에서는 현장 통용성을 반영하여 국내외적으로 가장 범용적으로 명명되고 있는 용어인 엔터티를 집합의 개념으로 지칭하고 인스턴스를 단수의 개념으로 명명하도록 한다.

데이터 모델의 핵심 요소인 이 세 가지를 이용하여 일정한 표기법에 의해 데이터 모델을 만들어 낼 수 있다. 다음은 다양한 표기법에 의해 생성되는 데이터 모델의 표기법을 설명한다.

8. 데이터 모델링의 이해관계자

가. 이해관계자의 데이터 모델링 중요성 인식

실제 업무시스템을 구축하는 실전 프로젝트에서는 데이터베이스를 전문적으로 하는 이른바 DBA(DataBase Administrator)가 데이터 모델링을 전적으로 하는 예는 거의 없다. 오히려 업무시스템을 개발하는 응용시스템 개발자가 데이터 모델링도 같이 하게 된다. 그 이유는 데이터 모델링이라는 과정이 단지 데이터베이스를 설계한다는 측면보다 업무를 이해하고 분석하여 표현하는 것이 중요하고, 표현된 내용을 바탕으로 프로젝트 관련자와 의사소통하고 프로그램이나 다른 표기법과 비교 검증하는 일을 수행하는 등 많은 시간을 업무를 분석하고 설계하는데 할애하기 때문에 업무영역별 개발팀에서 보통 데이터 모델링을 진행하게 되는 것이다.

물론 시스템이 대형화되면 모델링만을 전문적으로 담당하는 모델러를 투입하여 진행하는 경우도 있지만 이와 같은 경우도 실제 모델링 작업은 응용개발을 하는 사람이나 업무분석가(역할분담이 잘 되어 있을 경우)가 담당하고 모델러나 DBA 는 정확하게 모델링이 진행될 수 있도록 교육하고 제시하며 현안별로 직접 모델링을 진행하는 역할을 수행한다.

이와 같이 응용시스템을 개발하는 모든 시스템 엔지니어가 데이터 모델을 하거나 할 기회가 있음에도 불구하고 대부분의 사람들은 데이터 모델에 많은 관심을 갖지 않고 단지 프로그램을 개발하기

위한 프로그래밍 언어(Programming Language)에만 많은 관심을 두고 애플리케이션을 개발하는데에 훨씬 많은 시간을 투자하는 경우가 많다. 그러나 분명한 사실은 정보 시스템을 개발 한다고 할 때 데이터 모델링 데이터 베이스 구축 구축된 데이터의 적절한 활용은 다른 어떤 일(Task) 보다 중요하다는 점이다

- 대부분의 기업에 있어서 정보시스템의 데이터베이스 구조는 사용자에게 숨겨진 형태로 구축되어 왔다.

→ 정보의 고립화



- 프로그램은 6가지 유형의 데이터베이스 유지절차 - C. Finkelstein
- Programmer is the Navigator in the sea of data. - Bachmann

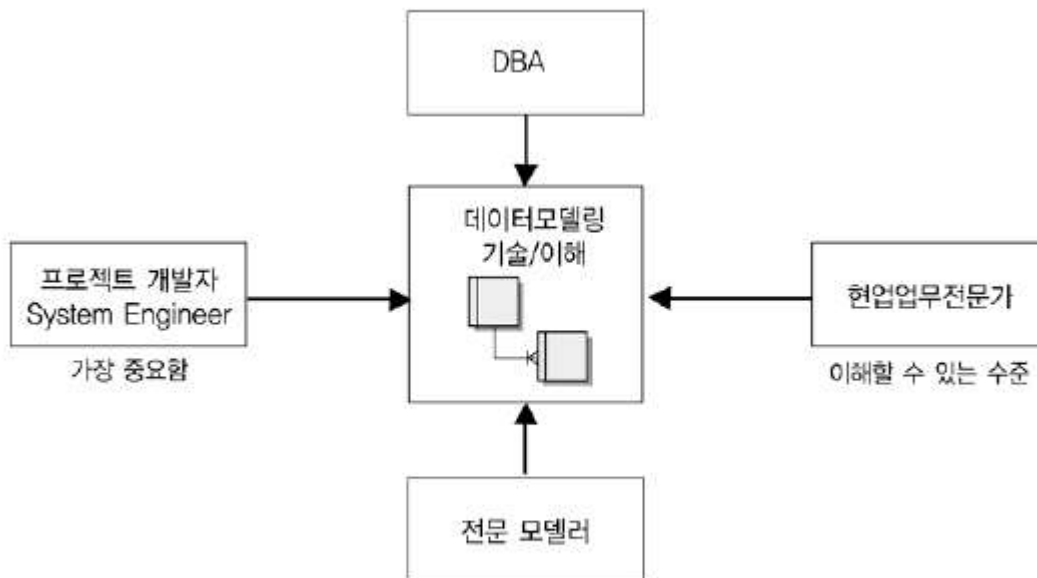
[그림 1-1-7] 자료 처리의 중심은 데이터

우리가 구축하려는 시스템 대부분을 데이터에 기반한, 데이터가 중심에 있는 정보시스템을 구축하기 때문에 정보시스템의 핵심에 있는 데이터베이스 설계를 잘못했을 때 미치는 영향력은 모든 프로그램, 시간에 따라 입력되는 모든 데이터, 그리고 그 데이터베이스에 발생하는 모든 트랜잭션에 영향을 미칠 수 밖에 없게 된다. Bachmann 은 ‘프로그래머는 데이터집합의 탐색자이다’ 라고 하였다. 그만큼 데이터에 대한 중요성을 높게 평가하는 것이다.

나. 데이터 모델링의 이해관계자

그러면 누가 데이터 모델링에 대해 연구하고 학습해야 하겠는가? 첫 번째는 정보시스템을 구축하는 모든 사람(전문적으로 코딩만하는 사람 포함)은 데이터 모델링도 전문적으로 할 수 있거나 적어도 완성된 모델을 정확하게 해석할 수 있어야 한다. 즉 프로젝트에 참여한 모든 IT 기술자들은 데이터 모델링에 대해 정확하게 알고 있어야 한다는 것을 의미한다.

두 번째는 IT 기술에 종사하거나 전공하지 않았더라도 해당 업무에서 정보화를 추진하는 위치에 있는 사람도 데이터 모델링에 대한 개념 및 세부사항에 대해 어느 정도 지식을 가지고 있어야 한다. 실제 프로젝트에서 보면 업무분석을 하는 도중에 현업의 업무담당자가 어느 사이에 데이터 모델링에 대해 상당한 이해를 하고 있음을 알게 된다. 그래야만 서로가 프로젝트 수행 중에 의사소통을 잘 할 수 있고 업무를 잘못 해석하여 잘못된 시스템을 구축하는 위험(Risk)을 줄일 수 있게 된다. 업무를 가장 잘 알고 있는 사람이 가장 훌륭한 모델러가 될 수 있다는 사실을 나타내는 예이다.



[그림 1-1-8] 데이터 모델 이해관계자

9. 데이터 모델의 표기법인 ERD의 이해

가. 데이터 모델 표기법

데이터 모델에 대한 표기법으로 1976년 피터첸(Peter Chen)이 Entity-relationship model(E-R Model)이라는 표기법을 만들었다. 엔터티를 사각형으로 표현하고 관계를 마름모 속성을 타원형으로 표현하는 이 표기법은 데이터 모델링에 대한 이론을 배울 때 많이 활용되고 있다. 데이터베이스 설계에 대해 우리나라 대학에서는 주로 이 Chen의 모델 표기법을 통해 배우고 있다. [표 1-1-6]은 엔터티와 속성 그리고 관계에 대한 다양한 표기법을 설명한 것이다

[표 1-1-6] 표기법

표기법	설명
<p>Chen</p>	<ul style="list-style-type: none"> - 대학교재에서 가장 많이 이용하는 표기법 - 실무적으로 사용안함
<p>IDEF1X</p>	<ul style="list-style-type: none"> - 마름모와 원을 이용한 표기법으로 실무현장에서는 소수 활용 - ERWin
<p>IE/Crow's Foot</p>	<ul style="list-style-type: none"> - 까마귀발 모양의 표기법으로 가장 많이 사용함 - ERWin, ERStudio
<p>Min-Max/ISO</p>	<ul style="list-style-type: none"> - 기수성을 좀 더 정교하게 표현한 방법으로 많이 활용안됨
<p>UML</p>	<ul style="list-style-type: none"> - 스테레오타입을 이용하여 엔티티 표현 - UML로 표현하여 데이터 모델링 할 때 사용 - Rational Rose
<p>Case*Method/Barker</p>	<ul style="list-style-type: none"> - Crow's Foot을 적용하면서 관계 표기법 등 일부 다름(Barker's Notation) - DA#

데이터아키텍처 전문가(DAP) 관련 자격에서는 바커(Barker) 표기법을 적용하여 설명했다면, 본 가이드에서는 범용적인 Information Engineering(이하 IE) 표기법과 바커 표기법을 모두 적용하여 설명을 진행하도록 한다. 표기법은 바커 표기법이든 IE 표기법이든 상호간에 기술적으로 전환이 가능하기 때문에 한 가지만 정확하게 알고 있어도 다른 표기법을 이해하는데 큰 어려움이 없을 것이다.

나. ERD(Entity Relationship Diagram) 표기법을 이용하여 모델링하는 방법

ERD는 각 업무분석에서 도출된 엔티티와 엔티티간의 관계를 이해하기 쉽게 도식화된 다이어그램으로 표시하는 방법으로서 실제 프로젝트에서는 도식화된 그림 정도로만 생각하지 않고 해당 업무에서 데이터의 흐름과 프로세스와의 연관성을 이야기하는 데 가장 중요한 표기법이자 산출물이다.

UML 표준 표기법을 사용하는 오브젝트 모델링에서는 궁극적으로 해당 업무에 가장 적절한 클래스 다이어그램을 그려내는 것이 가장 중요하다고 하면, 정보공학을 기반으로 하는 모델링에서는 해당 업무에 가장 적절한 ERD를 그려내는 것이 프로젝트의 지상과제이다. 오브젝트 모델링을 하더라도 관계형 데이터베이스를 대부분 사용하기 때문에 데이터베이스를 생성할 수 있는 데이터 모델 생성이 프로젝트에서 아주 중요한 TASK에 포함된다.

데이터분석이 어느 정도 완료되면 즉 엔티티, 관계, 속성 등이 데이터사전이나 각종 산출물에 의해 분석된 상태에서 ERD를 그리는 것이 원래 이론적인 작업 방법이지만, 실제 프로젝트에서는 분석된

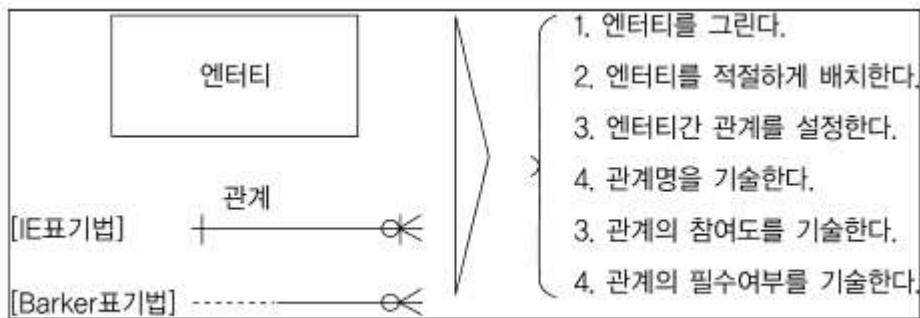
엔터티와 관계, 속성 정보가 바로 ERD에 표현되며 내부 프로젝트 인원이나 해당 업무고객과 대화할 때 핵심 업무산출물로 항상 이용된다.

ERD를 그리는 것은 물론 어떻게 그리든 업무에는 전혀 지장을 주지 않지만 일정한 규칙을 지정하여 그림으로써 데이터 모델을 누구나 공통된 시각으로 파악할 수 있고 의사소통을 원활하게 하는 장점이 있다.

여기에서 제시하는 방법은 가이드일 뿐이며 프로젝트 상황과 엔터티의 관련 순서에 따라 얼마든지 다르게 배치될 수 있음을 숙지하고 배치방법에 대한 원칙을 다음과 같이 설명한다. 최근에는 전문 데이터 모델링 툴을 활용하여 ERD를 그리게 되므로 다음과 같이 엔터티, 속성, 관계 순으로 진행하기 보다는 엔터티와 관계를 바로 표현하는 방식으로 진행하기도 한다. 다만 업무를 분석하여 무엇을 도출할 지에 대한 관점을 제시하고, 전문 데이터 모델링 툴이 없을 때 어떤 순서로 표현해야 하는지 알 수 있도록 방법을 설명한다.

1) ERD 작업순서 ERD를 작성하는 작업순서는 다음과 같다.

① 엔터티를 그린다. ② 엔터티를 적절하게 배치한다. ③ 엔터티간 관계를 설정한다. ④ 관계명을 기술한다. ⑤ 관계의 참여도를 기술한다. ⑥ 관계의 필수여부를 기술한다.



[그림 1-1-9] ERD 작업순서

ERD는 엔터티와 엔터티 사이의 관계가 있는 정보를 나타내므로 두 개를 이용하여 작성하고, 이에 따라 Primary Key와 Foreign Key를 ERD 규칙에 따라 기술하도록 한다. 엔터티는 사각형으로 표기하여 기술한다.

2) 엔터티 배치 엔터티를 처음에 어디에 배치하는지는 데이터 모델링 툴을 사용하든 사용하지 않던 중요한 문제이다. 일반적으로 사람의 눈은 왼쪽에서 오른쪽, 위 쪽에서 아래쪽으로 이동하는 경향이 있다. 따라서 데이터 모델링에서도 가장 중요한 엔터티를 왼쪽상단에 배치하고 이것을 중심으로 다른 엔터티를 나열하면서 전개하면 사람의 눈이 따라가기에 편리한 데이터 모델링을 전개할 수 있다. 해당 업무에서 가장 중요한 엔터티는 왼쪽 상단에서 조금 아래쪽 중앙에 배치하여 전체 엔터티와 어울릴 수 있도록 하면 향후 관계를 연결할 때 선이 꼬이지 않고 효과적으로 배치할 수 있게 된다.



[그림 1-1-10] 엔터티 배치방법

[그림 1-1-10]의 데이터 모델에서도 가장 중요한 엔터티인 고객과 주문을 왼쪽 상단에 배치하여 다른 엔터티를 연결하는 방식으로 엔터티를 배치하였다. 주문에 따라 출고가 이루어졌으므로 주문이 위에 출고가 아래에 위치해 있다. 두 번째 업무흐름에 중심이 되는 엔터티, 보통 업무 흐름에 있어서 중심이 되는 엔터티는 타 엔터티와 많은 관계를 가지고 있으므로 중앙에 배치하도록 한다. [그림 1-1-10]에서는 주문, 출고, 주문목록, 출고목록이 업무의 중심엔터티에 해당한다. 세 번째는 업무를 진행하는 중심엔터티와 관계를 갖는 엔터티들은 중심에 배치된 엔터티를 주위에 배치하도록 한다. [그림 1-1-10]에서는 창고, 고객, 사원, 재고가 이에 해당한다.

3) ERD 관계식서를 보고 서로 관련있는 엔터티간에 관계를 설정하도록 한다. 초기에는 모두 Primary Key 로 속성이 상속되는 식별자 관계를 설정하도록 한다. 중복되는 관계가 발생되지 않도록 하고 Circle 관계도 발생하지 않도록 유의하여 작성하도록 한다.



[그림 1-1-11] ERD 관계설정

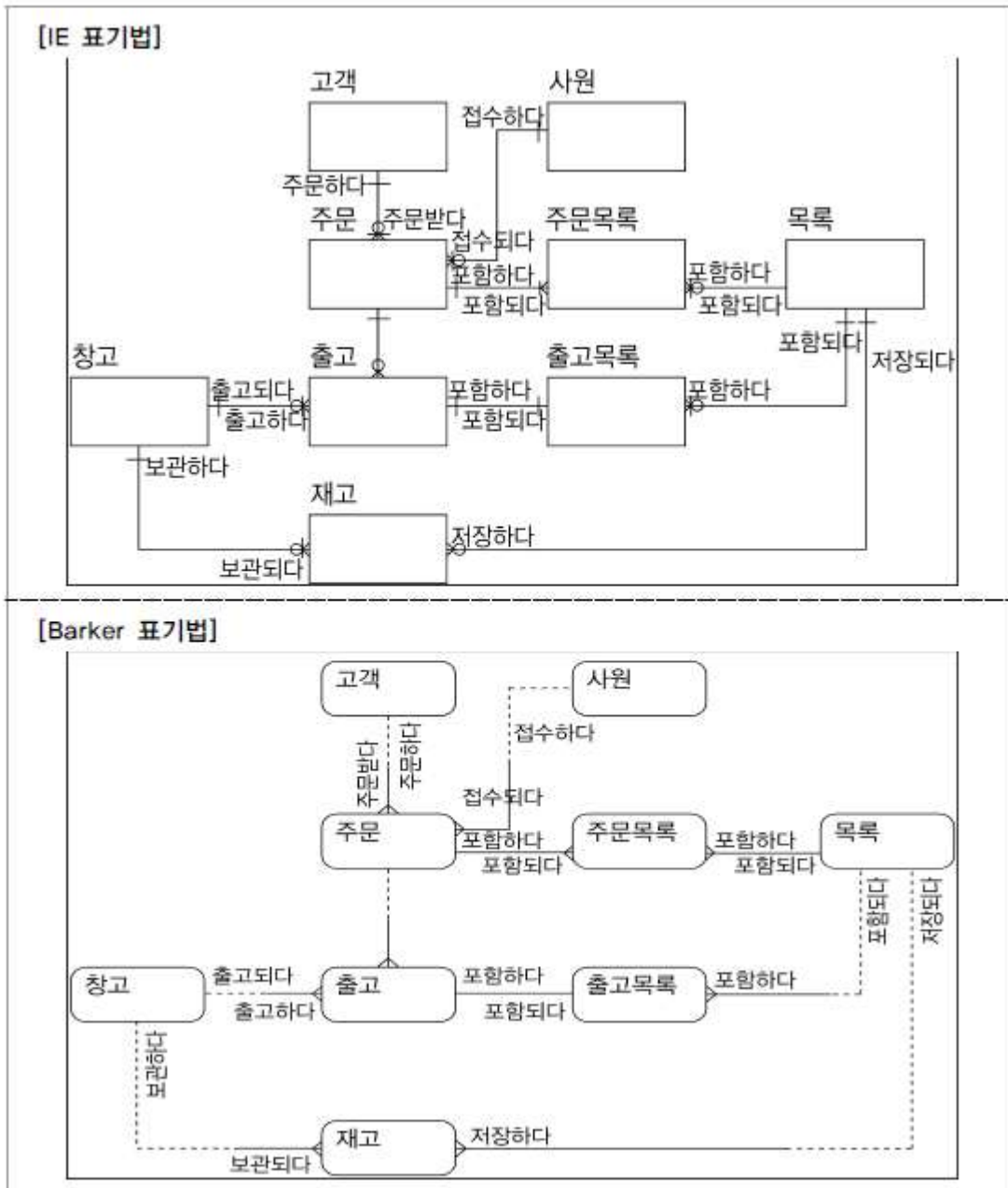
4) ERD 관계명의 표시 관계설정이 완료되면 연결된 관계에 관계이름을 부여하도록 한다. 관계이름은 현재형을 사용하고 지나치게 포괄적인 용어(예, 이다, 가진다 등)는 사용하지 않도록 한다.



[그림 1-1-12] ERD 관계명 표시

실제 프로젝트에서는 관계의 명칭을 크게 고려하지 않아도 무방하다. 왜냐하면 관계의 명칭이 나타나지 않아도 ERD의 흐름이 명확하게 드러나기 때문이다. 대부분의 관계는 엔터티의 성질과 주식별자를 보고 유추가 가능하다.

5) ERD 관계 관계차수와 선택성 표시 관계에 대한 이름을 모두 지정하였으면 관계가 참여하는 성격 중 엔터티내에 인스턴스들이 얼마나 관계에 참여하는 지를 나타내는 관계차수(Cardinality)를 표현한다. [그림 1-1-13]은 관계의 관계차수를 지정한 ERD의 모습을 보여준다. 관계설명에서도 언급하겠지만 IE 표기법으로는 하나(1, One)의 관계는 실선으로 표기하고 Barker 표기법으로는 점선과 실선을 혼합하여 표기한다. 다수참여(Many)의 관계는 까마귀발과 같은 모양으로 그려준다. 또한 관계의 필수/선택표시는 관계선에 원을 표현하여 ERD를 그리도록 한다.



[그림 1-1-13] 관계차수와 선택성 표시

10. 좋은 데이터 모델의 요소

일반적으로 시스템 구축 과정에서 생성되는 데이터 모델은 그 품질을 평가하는 것이 매우 어렵다. 사실 특정 데이터 모델이 업무 환경에서 요구하는 사항을 얼마나 잘 시스템적으로 구현할 수 있는가를 객관적으로 평가할 수 있다면 가장 좋은 평가의 방법일 것이다. 하지만 어디에도 이것을 객관적으로 평가할 수 있는 기준이 존재하지는 않는 것이 현실이다. 본 가이드에서는 이러한 상황에서 대체적으로 좋은 데이터 모델이라고 말할 수 있는 몇 가지의 요소들을 설명한다.

가. 완전성(Completeness)

업무에서 필요로 하는 모든 데이터가 데이터 모델에 정의되어 있어야 한다. 데이터 모델을 검증하기 위해서 가장 먼저 확인해야 할 부분이다. 이 기준이 충족되지 못하면 다른 어떤 평가 기준도 의미가 없어진다. 만약, 보험사의 데이터 모델에 고객의 직업을 관리하기 위한 속성이 존재하지 않는다면 어떨까? 이것은 심각한 데이터 모델의 문제점이다.

나. 중복배제(Non-Redundancy)

하나의 데이터베이스 내에 동일한 사실은 반드시 한 번만 기록하여야 한다. 예를 들면, 하나의 테이블에서 ‘나이’ 칼럼과 ‘생년월일’ 칼럼이 동시에 존재한다면 이것은 데이터 중복이라 볼 수 있다. 이러한 형태의 데이터 중복 관리로 인해서 여러 가지 바람직하지 않은 형태의 데이터 관리 비용을 지불할 수 있다. 예를 들면, 저장공간의 낭비, 중복 관리되고 있는 데이터의 일관성을 유지하기 위한 추가적인 데이터 조작 등이 대표적으로 낭비되는 비용이라고 볼 수 있다.

다. 업무규칙(Business Rules)

데이터 모델에서 매우 중요한 요소 중 하나가 데이터 모델링 과정에서 도출되고 규명되는 수많은 업무규칙(Business Rules)을 데이터 모델에 표현하고 이를 해당 데이터 모델을 활용하는 모든 사용자가 공유할 수 있도록 제공하는 것이다. 특히, 데이터 아키텍처에서 언급되는 논리 데이터 모델(Logical Data Model)에서 이러한 요소들이 포함되어야 함은 매우 중요하다.

예를 들면, 보험사의 직원들은 매월 여러 가지 항목에 대해서 급여를 지급받고 있고 이를 데이터로 관리하고 있다. 각 직원들은 월별로 하나 이상의 급여 항목(기본급, 상여금, 수당, 수수료, 등등)에 대해서 급여를 지급받는다. 여기에 더 나아가 각 직원은 직원 구분별(내근, 설계사, 계약직, 대리점 등)로 위의 급여 항목을 차등적으로 지급받는다는 업무규칙이 있다. 이러한 내용을 데이터 모델에 나타내야 한다. 이렇게 함으로써 해당 데이터 모델을 사용하는 모든 사용자(개발자, 관리자 등)가 해당 규칙에 대해서 동일한 판단을 하고 데이터를 조작할 수 있게 된다.

라. 데이터 재사용(Data Reusability)

데이터의 재사용성을 향상시키고자 한다면 데이터의 통합성과 독립성에 대해서 충분히 고려해야 한다. 과거에 정보시스템이 생성·운영된 형태를 되짚어 보면 철저하게 부서 단위의 정보시스템으로 설계되고 운용되어 왔다. 현재 대부분의 회사에서 진행하고 있는 신규 정보시스템의 구축 작업은 회사 전체 관점에서 공통 데이터를 도출하고 이를 전 영역에서 사용하기에 적절한 형태로 설계하여 시스템을 구축하게 된다. 이러한 형태의 데이터 설계에서 가장 중요하게 대두되는 것이 통합 모델이다. 통합 모델이어야만 데이터 재사용성을 향상시킬 수 있다.

또 한 측면에서 보면 과거 정보시스템의 데이터 구조의 가장 큰 특징은 데이터 모델이 별도로 존재하지 않고 애플리케이션의 부속품 정도로 인식되어져 왔던 것이 사실이다. 이러한 환경에서의 데이터는 프로세스의 흐름에 따라서 관리되게 마련이다. 이렇게 되면 데이터 중복이 많이 발생하고 데이터의 일관성 문제가 심각하게 초래된다. 데이터가 애플리케이션에 대해 독립적으로 설계되어야만 데이터 재사용성을 향상시킬 수 있다. 정보시스템은 비즈니스의 변화에 대해서 최적으로 적응하도록 끊임없이 요구된다. 하지만 일부 정보시스템의 데이터 모델은 이러한 변화에 대해서 현재의 데이터 구조를 거의 변화하지 않고도 변화에 대응할 수 있는 데이터 구조도 있을 것이고, 아주 적은 확장을 통해서 이러한 변화에 대응하는 것도 있을 것이다. 하지만, 이러한 변화에 대응하기 위해서 데이터 구조적으로 아주 많은 변화를 주어야만 한다면 변화의 대상이 되는 부분뿐만 아니라 정보시스템의 나머지 부분들도 많은 영향을 받게 될 것이다.

그래서 많은 기업들이 정보시스템을 구축하는 과정에서 데이터 구조의 확장성, 유연성에 많은 노력을 기울이고 있다. 결국 현대의 기업들이 동종의 타 기업으로부터 경쟁 우위에 자리매김하려고 한다면 구축하는 데이터 모델은 이러한 외부의 업무 환경 변화에 대해서 유연하게 대응할 수 있어야 한다. 특히 근래의 많은 패키지 시스템들이 가지고 있는 데이터 모델들은 확장성을 강조하기 위해서 많은 부분을 통합한 데이터 모델의 형태를 가지고 있다.

여기에서도 잘 나타나듯이 확장성을 담보하기 위해서는 데이터 관점의 통합이 불가피하다. 특히 정보시스템에서의 ‘행위의 주체’가 되는 집합의 통합, ‘행위의 대상’이 되는 집합의 통합, ‘행위 자

체'에 대한 통합 등은 전체 정보시스템의 안정성, 확장성을 좌우하는 가장 중요한 요소이다. 데이터 모델이 갖추어야 하는 중요한 요소 중에 하나는 기업이 관리하고자 하는 데이터를 합리적으로 균형이 있으면서도 단순하게 분류하는 것이다.

아무리 효율적으로 데이터를 잘 관리할 수 있다고 하더라도 그것의 사용, 관리 측면이 복잡하다면 잘 만들어진 데이터 모델이라고 할 수 없다. 동종의 비즈니스를 영위하는 기업이라 하더라도 각 회사의 데이터 모델을 비교해 보면 그 복잡도에는 많은 차이를 나타낸다. A 보험사는 계약 업무를 수행하기 위해서 10 개의 테이블을 정의하여 업무를 수행하는 반면에 B 회사는 100 개의 테이블을 정의하여 동일한 업무를 수행하고 있다. 두 회사의 데이터 모델의 차이점은 다음과 같다. 10 개의 테이블을 가지고 업무를 수행하고 있는 A 회사의 데이터 모델은 간결하지만 새로운 업무 환경의 변화에 대해서 확장성을 가지고 있다. B 회사는 겉으로는 새로운 업무 환경의 변화에 데이터 모델의 한계로 인해 테이블의 수가 지속적으로 증가해 왔다.

이렇게 됨으로써 데이터 모델은 간결하지 못하고 동일한 형태로 관리되어야 하는 데이터가 복잡한 형태로 관리되고 그들과의 관계를 가지고 있는 다른 여러 가지의 데이터들 또한 복잡한 형태의 관계들이 불가피 기본적인 전제는 통합이다. 합리적으로 잘 정돈된 방법으로 데이터를 통합하여 데이터의 집합을 정의하고 이를 데이터 모델로 잘 표현하여 활용한다면 웬만한 업무 변화에도 데이터 모델이 영향을 받지 않고 운용할 수 있게 된다.

마. 의사소통(Communication)

데이터 모델의 역할은 많다. 그 중에서도 중요한 것이 데이터 모델의 의사소통의 역할이다. 데이터 모델은 대상으로 하는 업무를 데이터 관점에서 분석하고 이를 설계하여 나오는 최종 산출물이다.

데이터를 분석 과정에서는 자연스럽게 많은 업무 규칙들이 도출된다. 이 과정에서 도출되는 많은 업무 규칙들은 데이터 모델에 엔터티, 서브타입, 속성, 관계 등의 형태로 최대한 자세하게 표현되어야 한다.

예를 들면, '사원' 테이블에는 어떠한 '사원구분'을 가지는 사원들이 존재하는지, '정규직', '임시직' 사원들이 같이 존재하는지, 아니면 또 다른 형태의 사원들이 존재하는지를 표현해야 한다. 더 나아가서 '호봉'이라는 속성은 '정규직'일 때에만 존재하는 속성인데 이러한 업무 규칙이 데이터 모델에 표현되어야 한다. 또한, 우리가 관리하는 사원들 중에서 '정규직' 사원들만이 '급여' 테이블과 관계를 가진다. 이러한 부분은 개별 관계로 데이터 모델에 표현되어야 한다.

이렇게 표현된 많은 업무 규칙들을 해당 정보시스템을 운용, 관리하는 많은 관련자들이 설계자가 정의한 업무 규칙들을 동일한 의미로 받아들이고 정보시스템을 활용할 수 있게 하는 역할을 하게 된다. 즉, 데이터 모델이 진정한 의사소통(Communication)의 도구로서의 역할을 하게 된다.

바. 통합성(Integration)

기업들이 과거부터 정보시스템을 구축해 왔던 방법은 개별 업무별로의 단위 정보시스템을 구축하여 현재까지 유지보수를 해오고 있는 것이 보통이다. 점진적인 확장과 보완의 방법으로 정보시스템을 구축해 왔기 때문에 동일한 성격의 데이터임에도 불구하고 전체 조직관점에서 보면 여러 곳에서 동일한 데이터가 존재하기 마련이다. 특히 이러한 데이터 중에서도 고객, 상품 등과 같이 마스터 성격의 데이터들이 분할되어 관리됨으로 인해 전체 조직 관점에서 데이터 품질, 관리, 활용 관점에서 많은 문제점들이 나타나고 있는 것이 현실이다.

가장 바람직한 데이터 구조의 형태는 동일한 데이터는 조직의 전체에서 한번만 정의되고 이를 여러 다른 영역에서 참조, 활용하는 것이다. 물론 이 때에 성능 등의 부가적인 목적으로 의도적으로 데이터를 중복시키는 경우는 존재할 수 있다. 동일한 성격의 데이터를 한 번만 정의하기 위해서는

공유 데이터에 대한 구조를 여러 업무 영역에서 공동으로 사용하기 용이하게 정의할 수 있어야 한다. 이러한 이유로 데이터 아키텍처의 중요성이 한층 더 부각되고 있다.

1. 엔터티의 개념

데이터 모델을 이해할 때 가장 명확하게 이해해야 하는 개념 중에 하나가 바로 엔터티(Entity)이다. 이것은 우리말로 실체, 객체라고 번역하기도 하는데 실무적으로 엔터티라는 외래어를 많이 사용하기 때문에 본 가이드에서는 엔터티라는 용어를 그대로 사용하기로 한다.

엔터티에 대해서 데이터 모델과 데이터베이스에 권위자가 정의한 사항은 다음과 같다.

- 변별할 수 있는 사물 - Peter Chen (1976) -
 - 데이터베이스 내에서 변별 가능한 객체 - C.J Date (1986) -
 - 정보를 저장할 수 있는 어떤 것 - James Martin (1989) -
 - 정보가 저장될 수 있는 사람, 장소, 물건, 사건 그리고 개념 등 - Thomas Bruce (1992) -
- 위 정의들의 공통점은 다음과 같다.
- 엔터티는 사람, 장소, 물건, 사건, 개념 등의 명사에 해당한다.
 - 엔터티는 업무상 관리가 필요한 관심사에 해당한다.
 - 엔터티는 저장이 되기 위한 어떤 것(Thing)이다.

엔터티란 “업무에 필요하고 유용한 정보를 저장하고 관리하기 위한 집합적인 것(Thing)”으로 설명할 수 있다. 또는, 엔터티는 업무 활동상 지속적인 관심을 가지고 있어야 하는 대상으로서 그 대상들 간에 동질성을 지닌 인스턴스들이나 그들이 행하는 행위의 집합으로 정의할 수 있다. 엔터티는 그 집합에 속하는 개체들의 특성을 설명할 수 있는 속성(Attribute)을 갖는데, 예를 들어 ‘학생’이라는 엔터티는 학번, 이름, 이수학점, 등록일자, 생일, 주소, 전화번호, 전공 등의 속성으로 특징지어질 수 있다. 이러한 속성 가운데에는 엔터티 인스턴스 전체가 공유할 수 있는 공통 속성도 있고, 엔터티 인스턴스 중 일부에만 해당하는 개별 속성도 있을 수 있다.

또한 엔터티는 인스턴스의 집합이라고 말할 수 있고, 반대로 인스턴스라는 것은 엔터티의 하나의 값에 해당한다고 정의할 수 있다. 예를 들어 과목은 수학, 영어, 국어가 존재할 수 있는데 수학, 영어, 국어는 각각이 과목이라는 엔터티의 인스턴스들이라고 할 수 있다. 또한 사건이라는 엔터티에는 사건번호 2010-001, 2010-002 등의 사건이 인스턴스가 될 수 있다. 엔터티를 이해할 때 눈에 보이는(Tangible)한 것만 엔터티로 생각해서는 안되며 눈에 보이지 않는 개념 등에도 엔터티로서 인식을 할 수 있어야 한다. 실제 업무상에는 눈에 보이지 않는 것(Thing)이 엔터티로 도출되는 경우가 많기 때문에 더욱더 주의할 필요가 있다.

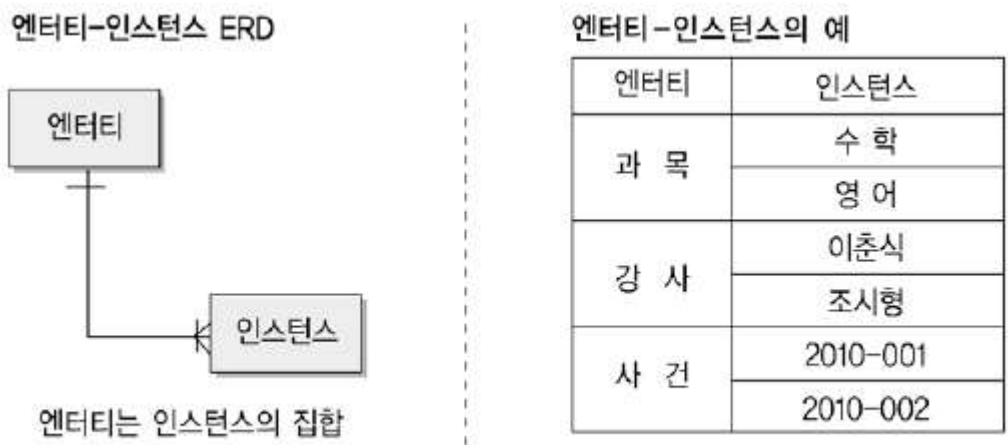


업무에 필요하고 유용한 정보를 저장하고 관리하기 위한 집합적인 것(thing)

[그림 1-1-14] 엔터티 종류

2. 엔터티와 인스턴스에 대한 내용과 표기법

엔터티를 표현하는 방법은 각각의 표기법에 따라 조금씩 차이는 있지만 대부분 사각형으로 표현된다. 다만 이 안에 표현되는 속성의 표현방법이 조금씩 다를 뿐이다. 엔터티와 엔터티간의 ERD를 그리면 (그림 1-1-15)와 같이 표현할 수 있다.



[그림 1-1-15] 엔터티와 인스턴스

(그림 1-1-15)에서 과목, 강사, 사건은 엔터티에 해당하고 수학, 영어는 과목이라는 엔터티의 인스턴스이고 이춘식, 조시형은 강사라는 엔터티의 인스턴스이며 사건번호인 2010-001, 2010-002는 사건 엔터티에 대한 인스턴스에 해당한다.

※ 참고 : 오브젝트 모델링에는 클래스(Class)와 오브젝트(Object)라는 개념이 있다. 클래스는 여러 개의 오브젝트를 포함하는 오브젝트 깡통이다. 이러한 개념은 정보공학의 엔터티가 인스턴스를 포함하는 개념과 비슷하다.

위의 엔터티와 인스턴스를 표현하면 (그림 1-1-16)과 같다.



[그림 1-1-16] 엔터티에 대한 표기법

3. 엔터티의 특징

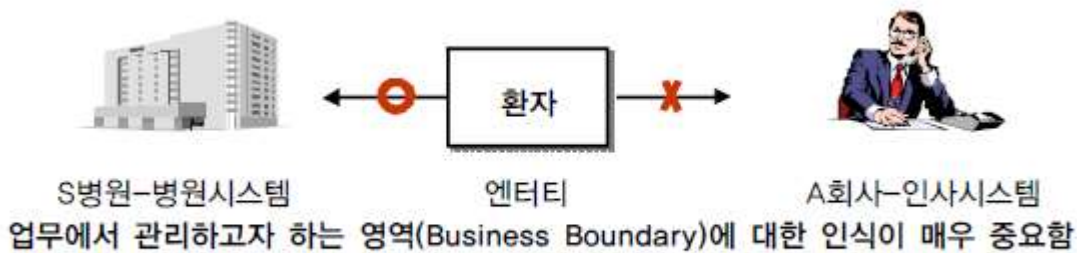
엔터티는 다음과 같은 특징을 가지고 있으며 만약 도출된 엔터티가 다음의 성질을 만족하지 못하면 적절하지 않은 엔터티일 확률이 높다.

- 반드시 해당 업무에서 필요하고 관리하고자 하는 정보이어야 한다.(예. 환자, 토익의 응시횟수, ...)
- 유일한 식별자에 의해 식별이 가능해야 한다.

- 영속적으로 존재하는 인스턴스의 집합이어야 한다.('한 개'가 아니라 '두 개 이상')
- 엔터티는 업무 프로세스에 의해 이용되어야 한다.
- 엔터티는 반드시 속성이 있어야 한다.
- 엔터티는 다른 엔터티와 최소 한 개 이상의 관계가 있어야 한다.

가. 업무에서 필요로 하는 정보

엔터티 특징의 첫 번째는 반드시 시스템을 구축하고자 하는 업무에서 필요로 하고 관리하고자 하는 정보여야 한다는 점이다. 예를 들어 환자라는 엔터티는 의료시스템을 개발하는 병원에서는 반드시 필요한 엔터티이지만 일반회사에서 직원들이 병에 걸려 업무에 지장을 준다하더라도 이 정보를 그 회사의 정보로서 활용하지는 않을 것이다. 즉 시스템 구축 대상인 해당업무에서 그 엔터티를 필요로 하는가를 판단하는 것이 중요하다.



[그림 1-1-17] 엔터티 특성-필요성

사람이 살아가면서 환자는 발생할 수 밖에 없다. 그러나 일반회사의 인사시스템에서는 비록 직원들에 의해서 환자가 발생이 되지만 인사업무 영역에서 환자를 별도로 관리할 필요가 없다. 다른 예로 병원에서는 환자가 해당 업무의 가장 중요한 엔터티가 되어 꼭 관리해야 할 엔터티가 된다. 이와 같이 엔터티를 도출할 때는 업무영역 내에서 관리할 필요가 있는지를 먼저 판단하는 것이 중요하다.

나. 식별이 가능해야 함

두 번째는 식별자(Unique Identifier)에 의해 식별이 가능해야 한다는 점이다. 어떤 엔터티이건 임의의 식별자(일련번호)를 부여하여 유일하게 만들 수는 있지만, 엔터티를 도출하는 경우에 각각의 업무적으로 의미를 가지는 인스턴스가 식별자에 의해 한 개씩만 존재하는지 검증해 보아야 한다. 유일한 식별자는 그 엔터티의 인스턴스만의 고유한 이름이다. 두 개 이상의 엔터티를 대변하면 그 식별자는 잘못 설계된 것이다. 예를 들어 직원을 구분할 수 있는 방법은 이름이나 사원번호가 될 수가 있다. 그러나 이름은 동명이인(同名異人)이 될 수 있으므로 유일하게 식별될 수 없다. 사원번호는 회사에 입사한 사람에게 고유하게 부여된 번호이므로 유일한 식별자가 될 수 있는 것이다.



인스턴스 각각을 구분하기 위한 유일한 식별자가 존재해야 함

[그림 1-1-18] 엔터티 특성-유일성

다. 인스턴스의 집합

세 번째는 연속적으로 존재하는 인스턴스의 집합이 되어야 한다는 점이다. 엔터티의 특징 중 “한 개”가 아니라 “두 개 이상”이라는 집합개념은 매우 중요한 개념이다. 두 개 이상이라는 개념은 엔터티뿐만 아니라 엔터티간의 관계, 프로세스와의 관계 등 업무를 분석하고 설계하는 동안 설계자가 모든 업무에 대입해보고 검증해 보아야 할 중요한 개념이다. 하나의 엔터티는 여러 개의 인스턴스를 포함한다.

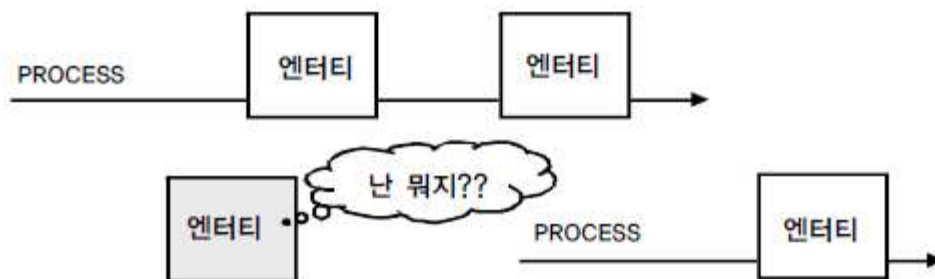


인스턴스가 한 개 밖에 없는 회사, 병원 엔터티는 집합이 아니므로 엔터티 성립이 안됨

[그림 1-1-19] 엔터티 특성-인스턴스 수

라. 업무프로세스에 의해 이용

네 번째는 업무프로세스(Business Process)가 그 엔터티를 반드시 이용해야 한다는 점이다. 첫 번째 정의에서처럼 업무에서 반드시 필요하다고 생각하여 엔터티로 선정하였는데 업무프로세스에 의해 전혀 이용되지 않는다면 업무 분석이 정확하게 안되어 엔터티가 잘못 선정되거나 업무프로세스 도출이 적절하게 이루어지지 않았음을 의미한다. 이러한 경우는 데이터 모델링을 할 때 미처 발견하지 못하다가 프로세스 모델링을 하면서 데이터 모델과 검증을 하거나, 상관 모델링을 할 때 엔터티와 단위프로세스를 교차 점검하면서 문제점이 도출된다.



업무 프로세스에 의해 이용되지 않는 엔터티는 그 업무의 엔터티가 아님

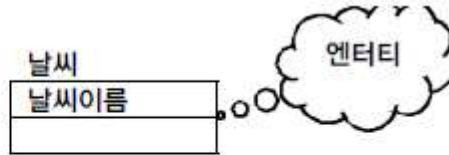
[그림 1-1-20] 엔터티 특성-프로세스 존재

[그림 1-1-20]과 같이 업무프로세스에 의해 CREATE, READ, UPDATE, DELETE 등이 발생하지 않는 고립된 엔터티의 경우는 엔터티를 제거하거나 아니면 누락된 프로세스가 존재하는지 살펴보고 해당 프로세스를 추가해야 한다.

마. 속성을 포함

다섯 번째는 엔터티에는 반드시 속성(Attributes)이 포함되어야 한다는 점이다. 속성을 포함하지 않고 엔터티의 이름만 가지고 있는 경우는 관계가 생략되어 있거나 업무 분석이 미진하여 속성정보가 누락되는 경우에 해당한다. 또한 주식별자만 존재하고 일반속성은 전혀 없는 경우도 마찬가지로 적절한 엔터티라고 할 수 없다. 단, 예외적으로 관계엔터티(Associative Entity)의 경우는 주식별자 속성만 가지고 있어도 엔터티로 인정한다.

태풍
태풍이름
발생지역
풍속

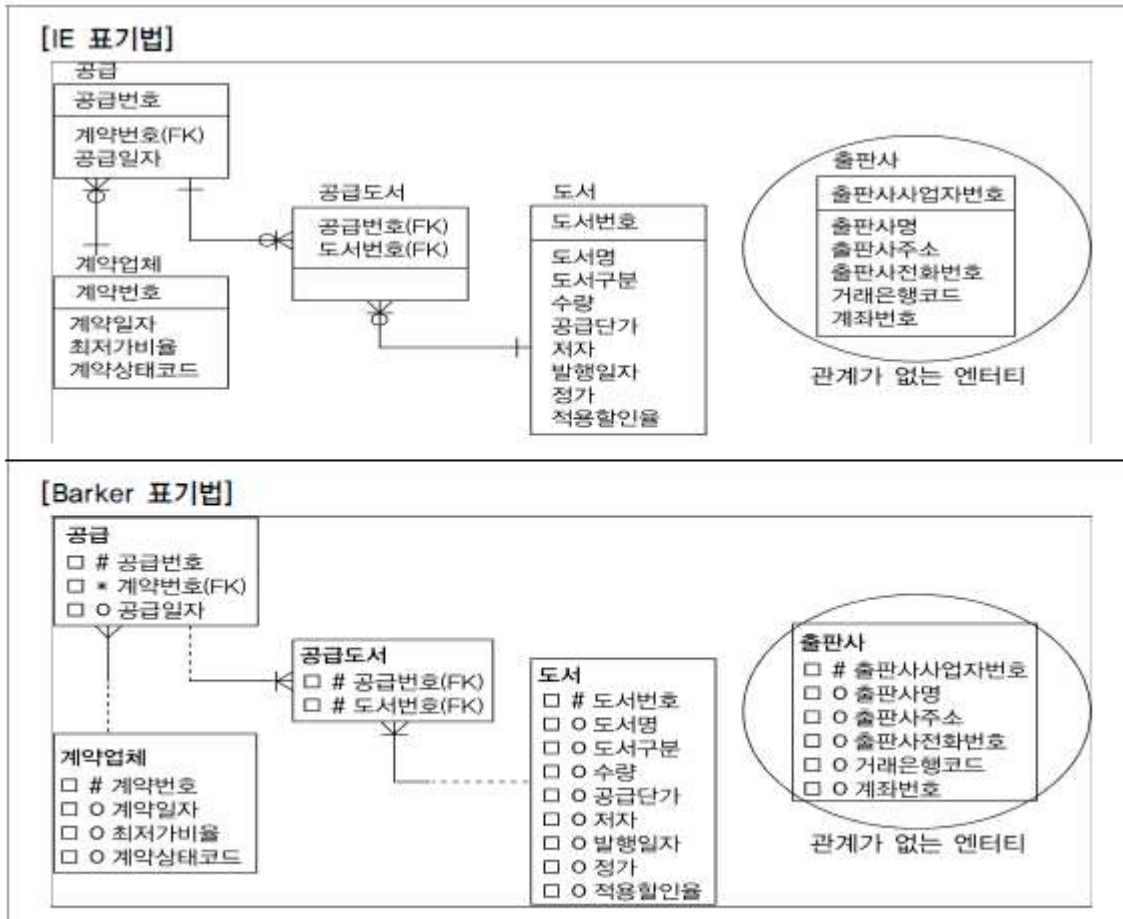


속성이 존재하지 않는 오브젝트는 엔터티가 될 수 없음

[그림 1-1-21] 엔터티 특성-속성의 존재

바. 관계의 존재

여섯 번째는 엔터티는 다른 엔터티와 최소 한 개 이상의 관계가 존재해야 한다는 것이다. 기본적으로 엔터티가 도출되었다는 것은 해당 업무내에서 업무적인 연관성(존재적 연관성, 행위적 연관성)을 가지고 다른 엔터티와의 연관의 의미를 가지고 있음을 나타낸다. 그러나 관계가 설정되지 않은 엔터티의 도출은 부적절한 엔터티가 도출되었거나 아니면 다른 엔터티와 적절한 관계를 찾지 못했을 가능성이 크다.



엔터티가 관계가 없으면, 잘못된 엔터티이거나 관계가 누락되었을 가능성이 큼

[그림 1-1-22] 엔터티 특성-관계의 존재

단, 데이터 모델링을 하면서 관계를 생략하여 표현해야 하는 경우는 다음과 같은 통계성 엔터티 도출, 코드성 엔터티 도출, 시스템 처리시 내부 필요에 의한 엔터티 도출과 같은 경우이다.

- 1) 통계를 위한 엔터티의 경우는 업무진행 엔터티로부터 통계업무만(Read Only)을 위해 별도로 엔터티를 다시 정의하게 되므로 엔터티간의 관계가 생략되는 경우에 해당한다.
- 2) 코드를 위한 엔터티의 경우 너무 많은 엔터티와 엔터티간의 관계 설정으로 인해 데이터 모델의 읽기효율성(Readability)이 저하되어 도저히 모델링 작업을 진행할 수 없게 된다. 또한 코드성 엔터

티는 물리적으로 테이블과 프로그램 구현 이후에도 외부키에 의한 참조무결성을 체크하기 위한 규칙을 데이터베이스 기능에 맡기지 않는 경우가 대부분이기 때문에 논리적으로나 물리적으로 관계를 설정할 이유가 없다.

3) 시스템 처리시 내부 필요에 의한 엔터티(예를 들어, 트랜잭션 로그 테이블 등)의 경우 트랜잭션이 업무적으로 연관된 테이블과 관계 설정이 필요하지만 이 역시 업무적인 필요가 아니고 시스템 내부적인 필요에 의해 생성된 엔터티이므로 관계를 생략하게 된다.

4. 엔터티의 분류

엔터티는 엔터티 자신의 성격에 의해 실체유형에 따라 구분하거나 업무를 구성하는 모습에 따라 구분이 되는 발생시점에 의해 분류해 볼 수 있다.

가. 유무(有無)형에 따른 분류

일반적으로 엔터티는 유무형에 따라 유형엔터티, 개념엔터티, 사건엔터티로 구분된다.

유형엔터티(Tangible Entity)는 물리적인 형태가 있고 안정적이며 지속적으로 활용되는 엔터티로 업무로부터 엔터티를 구분하기가 가장 용이하다. 예를 들면, 사원, 물품, 강사 등이 이에 해당된다. 개념엔터티(Conceptual Entity)는 물리적인 형태는 존재하지 않고 관리해야 할 개념적 정보로 구분이 되는 엔터티로 조직, 보험상품 등이 이에 해당된다.

사건 엔터티(Event Entity)는 업무를 수행함에 따라 발생하는 엔터티로서 비교적 발생량이 많으며 각종 통계자료에 이용될 수 있다. 주문, 청구, 미납 등이 이에 해당된다.

나. 발생시점(發生時點)에 따른 분류

엔터티의 발생시점에 따라 기본/키엔터티(Fundamental Entity, Key Entity), 중심엔터티(Main Entity), 행위엔터티(Active Entity)로 구분할 수 있다.

1) 기본엔터티

기본엔터티란 그 업무에 원래 존재하는 정보로서 다른 엔터티와 관계에 의해 생성되지 않고 독립적으로 생성이 가능하고 자신은 타 엔터티의 부모의 역할을 하게 된다. 다른 엔터티로부터 주식별자를 상속받지 않고 자신의 고유한 주식별자를 가지게 된다. 예를 들어 사원, 부서, 고객, 상품, 자재 등이 기본엔터티가 될 수 있다.

2) 중심엔터티

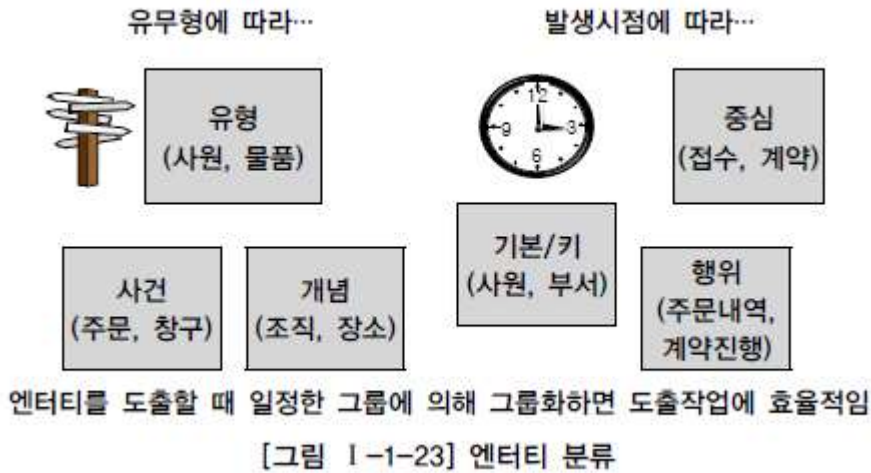
중심엔터티란 기본엔터티로부터 발생되고 그 업무에 있어서 중심적인 역할을 한다. 데이터의 양이 많이 발생되고 다른 엔터티와의 관계를 통해 많은 행위엔터티를 생성한다. 예를 들어 계약, 사고, 예금원장, 청구, 주문, 매출 등이 될 수 있다.

3) 행위엔터티

행위엔터티는 두 개 이상의 부모엔터티로부터 발생되고 자주 내용이 바뀌거나 데이터량이 증가된다. 분석초기 단계에서는 잘 나타나지 않으며 상세 설계단계나 프로세스와 상관모델링을 진행하면서 도출될 수 있다. 예를 들어 주문목록, 사원변경이력 등이 포함된다.

다. 엔터티 분류 방법의 예

[그림 1-1-23]은 두 가지 엔터티 분류 방법에 대한 예를 나타낸 것이다.



이 밖에도 엔터티가 스스로 생성될 수 있는지 여부에 따라 독립엔터티인지 의존엔터티인지를 구분할 수도 있다.

5. 엔터티의 명명

엔터티를 명명하는 일반적인 기준은 용어를 사용하는 모든 표기법이 다 그렇듯이 첫 번째는 가능하면 현업업무에서 사용하는 용어를 사용한다. 두 번째는 가능하면 약어를 사용하지 않는다. 세 번째는 단수명사를 사용한다. 네 번째는 모든 엔터티에서 유일하게 이름이 부여되어야 한다. 다섯 번째는 엔터티 생성의미대로 이름을 부여한다.

첫 번째에서 네 번째에 해당하는 원칙은 대체적으로 잘 지켜진다. 그러나 다섯 번째 원칙인 “엔터티 생성의미대로 이름을 부여한다.”에 대해서는 적절하지 못한 엔터티명이 부여되는 경우가 빈번하게 발생한다. 중심엔터티에서도 간혹 적절하지 못한 엔터티명을 사용한 경우가 발생되고 행위엔터티의 경우에는 꽤 많은 경우에 적절하지 못한 엔터티명을 사용하는 경우가 발생된다. 예를 들어, 고객이 어떤 제품들에 대해 주문하여 발생하는 행위엔터티에 대해 주문목록이라고도 할 수 있고 고객제품이라고 할 수 있다. 만약 고객제품이라고 하면 ‘고객이 주문한 제품’인지 아니면 ‘고객의 제품’인지 의미가 애매모호해질 수 있게 된다. 엔터티의 이름을 업무목적에 따라 생성되는 자연스러운 이름을 부여해야 하는데 이와 상관없이 임의로 이름을 부여하게 되면 프로젝트에서는 커뮤니케이션 오류로 인해 문제를 야기할 수 있게 된다.

1. 속성 (Attribute)의 개념

속성이란 사전적인 의미로는 사물(事物)의 성질, 특징 또는 본질적인 성질, 그것이 없다면 실체를 생각할 수 없는 것으로 정의할 수 있다. 본질적 속성이란 어떤 사물 또는 개념에 없어서는 안 될 징표(徵表)의 전부이다. 이 징표는 사물이나 개념이 어떤 것인지를 나타내고 그것을 다른 것과 구별하는 성질이라고 할 수 있다.

이런 사전적인 정의 이외에 데이터 모델링 관점에서 속성을 정의하자면, “업무에서 필요로 하는 인스턴스로 관리하고자 하는 의미상 더 이상 분리되지 않는 최소의 데이터 단위”로 정의할 수 있다. 업무상 관리하기 위한 최소의 의미 단위로 생각할 수 있고, 이것은 엔터티에서 한 분야를 담당하고 있다.



강사

엔터티는 속성들에 의해 설명된다.

속성들

- 이름
- 주소
- 생년월일
- 계약일자
- 전문분야

속성은 업무에서 필요로 하는 인스턴스에서 관리하고자 하는
의미상 더 이상 분리되지 않는 최소의 데이터 단위

[그림 1-1-24] 속성의 정의

속성의 정의를 정리해 보면 다음과 같다.

- 업무에서 필요로 한다.
- 의미상 더 이상 분리되지 않는다.
- 엔터티를 설명하고 인스턴스의 구성요소가 된다.

의미상 더 이상 분리되지 않는다는 특징을 살펴보면 다음과 같다. 예를 들어 생년월일은 하나로서 의미가 있다. 만약 이것을 생년, 생월, 생일로 구분한다면 이것은 사실상 하나의 속성을 관리목적에 따라 구분했다라고 이해할 수 있다. 이러한 이유로 인해 S/W 비용을 산정하는 기능점수(Function Point)를 산정할 때 이렇게 분리된 속성은 하나의 속성(DET)으로 계산하게 된다. 그러나 만약 서로 관련이 없는 이름, 주소를 하나의 속성 ‘이름주소’로 정의하면 어떻게 될까? 이것은 하나의 속성의 두 개의 의미를 갖기 때문에 기본속성으로서 성립하지 않게 된다. 이렇게 정리된 속성은 그냥 값의 의미로서 속성보다는 내역(Description)으로서 속성으로 예를 들어 인적사항이라는 속성으로 정의하여 관리할 수는 있다.

2. 엔터티, 인스턴스와 속성, 속성값에 대한 내용과 표기법

가. 엔터티, 인스턴스, 속성, 속성값의 관계

엔터티에는 두 개 이상의 인스턴스가 존재하고 각각의 엔터티에는 고유의 성격을 표현하는 속성정보를 두 개 이상 갖는다. 업무에서는 엔터티를 구성하는 특징이 무엇인지 또한 각각의 인스턴스들은 어떤 성격의 데이터로 구성되는지를 파악하는 작업이 필요하다. 분석단계에서 엔터티 내에 존재하는 여러 개의 인스턴스가 가지는 동일한 성격은 무엇인지를 파악하고 이에 이름을 부여하여 엔터티의 속성으로 기술하는 작업이 필요하다.

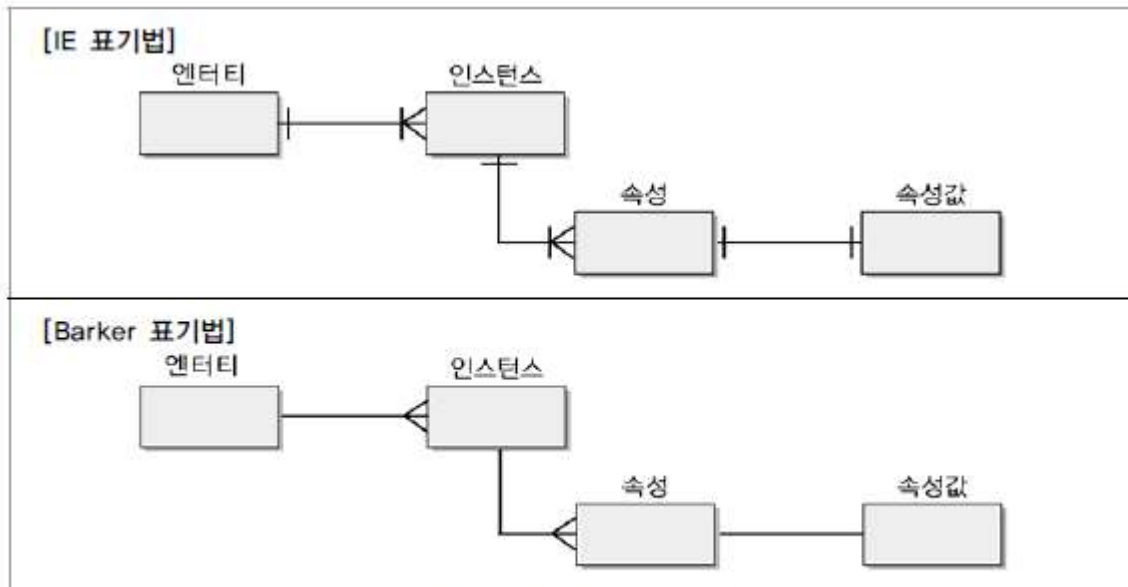
예를 들면 사원은 이름, 주소, 전화번호, 직책 등을 가질 수 있다. 사원이라는 엔터티에 속한 인스턴스들의 성격을 구체적으로 나타내는 항목이 바로 속성이다. 각각의 인스턴스는 속성의 집합으로 설명될 수 있다. 하나의 속성은 하나의 인스턴스에만 존재할 수 있다. 속성은 관계로 기술될 수 없고

자신이 속성을 가질 수도 없다.

엔터티 내에 있는 하나의 인스턴스는 각각의 속성들의 대해 한 개의 속성값만을 가질 수 있다. 예를 들면 사원의 이름은 홍길동이고 주소는 서울시 강남구이며, 전화번호는 123-4567, 직책은 대리이다. 이름, 주소, 전화번호, 직책은 속성이고 홍길동, 서울시 강남구, 123-4567, 대리는 속성값이다. 그러므로 속성값은 각각의 엔터티가 가지는 속성들의 구체적인 내용이라 할 수 있다.

엔터티, 인스턴스, 속성, 속성값에 대한 관계를 분석하면 다음과 같은 결과를 얻을 수 있다.

- 한 개의 엔터티는 두 개 이상의 인스턴스의 집합이어야 한다.
- 한 개의 엔터티는 두 개 이상의 속성을 갖는다.
- 한 개의 속성은 한 개의 속성값을 갖는다.



[그림 1-1-25] 엔터티-속성의 관계

속성은 엔터티에 속한 엔터티에 대한 자세하고 구체적인 정보를 나타내며 각각의 속성은 구체적인 값을 갖게 된다.

예를 들어 사원이라는 엔터티에는 홍길동이라는 사람(엔터티)이 있을 수 있다. 홍길동이라는 사람의 이름은 홍길동이고 주소는 서울시 강서구이며 생년월일 1967년 12월 31일이다. 여기에 이름, 주소, 생년월일과 같은 각각의 값을 대표하는 이름들을 속성이라 하고 홍길동, 서울시 강서구, 1967년 12월 31일과 같이 각각의 이름에 대한 구체적인 값을 속성 값(VALUE)이라고 한다.

나. 속성의 표기법

속성의 표기법은 엔터티 내에 이름을 포함하여 표현하면 된다.



[그림 1-1-26] 속성의 표기법

3. 속성의 특징

속성은 다음과 같은 특징을 가지고 있으며 만약 도출된 속성이 다음의 성질을 만족하지 못하면 적절하지 않은 속성일 확률이 높다.

- 엔터티와 마찬가지로 반드시 해당 업무에서 필요하고 관리하고자 하는 정보이어야 한다. (예, 강사의 교재이름)
- 정규화 이론에 근간하여 정해진 주식별자에 함수적 종속성을 가져야 한다.
- 하나의 속성에는 한 개의 값만을 가진다. 하나의 속성에 여러 개의 값이 있는 다중값일 경우 별도의 엔터티를 이용하여 분리한다.

4. 속성의 분류

가. 속성의 특성에 따른 분류

속성은 업무분석을 통해 바로 정의한 속성을 기본속성(Basic Attribute), 원래 업무상 존재하지는 않지만 설계를 하면서 도출해내는 속성을 설계속성(Designed Attribute), 다른 속성으로부터 계산이나 변형이 되어 생성되는 속성을 파생속성(Derived Attribute)이라고 한다.

1) 기본속성

기본 속성은 업무로부터 추출한 모든 속성이 여기에 해당하며 엔터티에 가장 일반적이고 많은 속성을 차지한다. 코드성 데이터, 엔터티를 식별하기 위해 부여된 일련번호, 그리고 다른 속성을 계산하거나 영향을 받아 생성된 속성을 제외한 모든 속성은 기본속성이다. 주의해야 할 것은 업무로부터 분석한 속성이라도 이미 업무상 코드로 정의한 속성이 많다는 것이다. 이러한 경우도 속성의 값이 원래 속성을 나타내지 못하므로 기본속성이 되지 않는다.

2) 설계속성

설계속성은 업무상 필요한 데이터 이외에 데이터 모델링을 위해, 업무를 규칙화하기 위해 속성을 새로 만들거나 변형하여 정의하는 속성이다. 대개 코드성 속성은 원래 속성을 업무상 필요에 의해 변형하여 만든 설계속성이고 일련번호와 같은 속성은 단일(Unique)한 식별자를 부여하기 위해 모델 상에서 새로 정의하는 설계속성이다.

3) 파생속성

파생속성은 다른 속성에 영향을 받아 발생하는 속성으로서 보통 계산된 값들이 이에 해당된다. 다

른 속성에 영향을 받기 때문에 프로세스 설계 시 데이터 정합성을 유지하기 위해 유의해야 할 점이 많으며 가급적 파생속성을 적게 정의하는 것이 좋다.



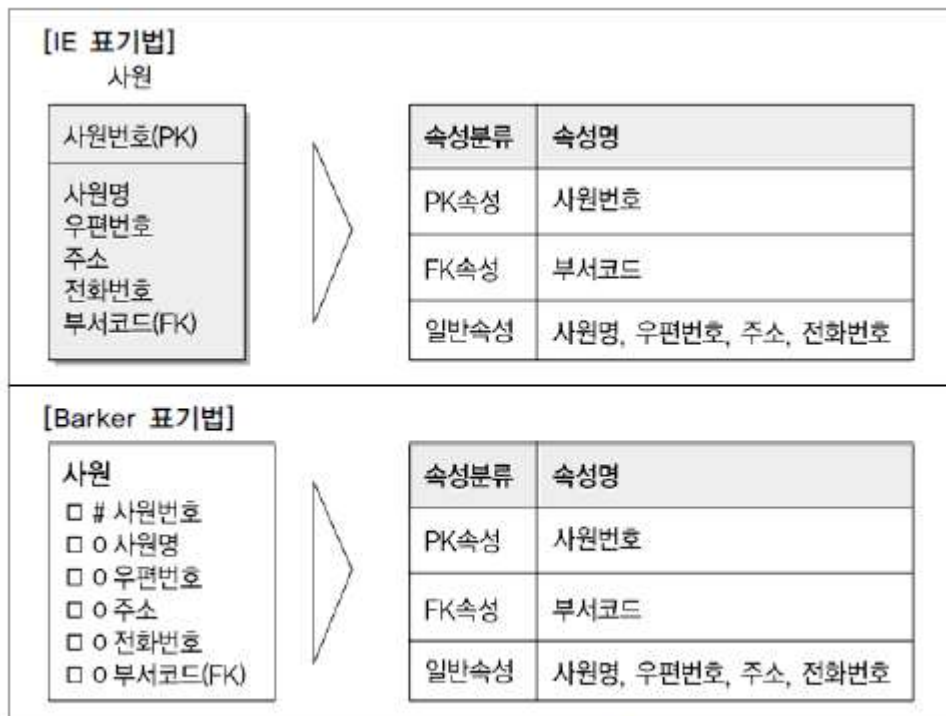
이러한 분류 방식은 프로젝트에서 엄격하게 분류하여 속성정의서에 나열하는 경우도 있다. 이 때 파생속성의 경우는 그 값이 계산된 로직을 속성의 정의서의 기록함으로써 향후 속성 값에 대한 검증 시 활용되기도 한다. 파생속성은 그 속성이 가지고 있는 계산방법에 대해서 반드시 어떤 엔터티에 어떤 속성에 의해 영향을 받는지 정의가 되어야 한다.

예를 들어 ‘이자’라는 속성이 존재한다고 하면 이자는 원금이 1,000 원이고 예치기간이 5 개월이며 이자율이 5.0%에서 계산되는 속성값이다. 그렇다면 이자는 원금이 1,000 원에서 2,000 원으로 변하여도 영향을 받고 예치기간이 5 개월에서 7 개월로 증가하여도 값이 변하며 이자율이 5.0%에서 6.0%로 되어도 이자속성이 가지는 값은 변할 것이다. 한 번 값이 변해도 또 다시 영향을 미치는 속성값의 조건이 변한다면 이자의 값은 지속적으로 변경될 것이다.

이와 같이 타 속성에 의해 지속적으로 영향을 받아 자신의 값이 변하는 성질을 가지고 있는 속성이 파생속성이다. 파생속성은 될 수 있으면 꼭 필요한 경우에만 정의하도록 하여 업무로직이 속성내부에 숨지 않도록 하는 것이 좋다. 파생속성을 정의한 경우는 속성정의서에 파생속성이 가지는 업무로직을 기술하여 데이터의 정합성이 유지될 수 있도록 해야 하며 그 파생속성에 원인이 되는 속성을 이용하는 모든 애플리케이션에서는 값을 생성하고, 수정하고 삭제할 때 파생속성도 함께 고려해 주어야 한다. 파생속성은 일반 엔터티에서는 많이 사용하지 않으며 통계관련 엔터티나 배치 작업이 수행되면서 발생하는 엔터티의 경우 많이 이용된다.

나. 엔터티 구성방식에 따른 분류

엔터티를 식별할 수 있는 속성을 PK(Primary Key)속성, 다른 엔터티와의 관계에서 포함된 속성을 FK(Foreign Key)속성, 엔터티에 포함되어 있고 PK, FK에 포함되지 않은 속성을 일반속성이라 한다.



[그림 1-1-28] 속성의 분류

또한 속성은 그 안에 세부 의미를 쪼갤 수 있는지에 따라 단순형 혹은 복합형으로 분류할 수 있다. 예를 들면 주소 속성은 시, 구, 동, 번지 등과 같은 여러 세부 속성들로 구성될 수 있는데 이를 복합 속성(Composite Attribute)이라 한다. 또한 나이, 성별 등의 속성은 더 이상 다른 속성들로 구성될 수 없는 단순한 속성이므로 단순 속성(Simple Attribute)이라 한다.

일반적으로 속성은 하나의 값을 가지고 있으나, 그 안에 동일한 성질의 여러 개의 값이 나타나는 경우가 있다. 이 때 속성 하나에 한 개의 값을 가지는 경우를 단일값(Single Value), 그리고 여러 개의 값을 가지는 경우를 다중값(Multi Value) 속성이라 한다. 주민등록번호와 같은 속성은 반드시 하나의 값만 존재하므로 이 속성은 단일값 속성(Single-Valued Attribute)이라 하고, 어떤 사람의 전화번호와 같은 속성은 집, 휴대전화, 회사 전화번호와 같이 여러 개의 값을 가질 수 있다. 자동차의 색상 속성도 차 지붕, 차체, 외부의 색이 다를 수 있다. 이런 속성을 다중값 속성(Multi-Valued Attribute)이라 한다. 다중값 속성의 경우 하나의 엔터티에 포함될 수 없으므로 1차 정규화를 하거나, 아니면 별도의 엔터티를 만들어 관계로 연결해야 한다.

5. 도메인(Domain)

각 속성은 가질 수 있는 값의 범위가 있는데 이를 그 속성의 도메인(Domain)이라 한다. 예를 들면 학생이라는 엔터티가 있을 때 학점이라는 속성의 도메인은 0.0에서 4.0 사이의 실수 값이며 주소라는 속성은 길이가 20 자리 이내인 문자열로 정의할 수 있다. 여기서 물론 각 속성은 도메인 이외의 값을 갖지 못한다. 따라서 도메인을 좀더 이해하기 쉽게 정리하면, 엔터티 내에서 속성에 대한 데이터타입과 크기 그리고 제약사항을 지정하는 것이라 할 수 있다.

6. 속성의 명명(Naming)

C/S(Client/Server) 환경이든 Web 환경이든 속성명이 곧 사용자 인터페이스(User Interface)에 나타나기 때문에 업무와 직결되는 항목이다. 그래서 속성 이름을 정확하게 부여하고 용어의 혼란을 없애기 위해 용어사전이라는 업무사전을 프로젝트에 사용하게 된다. 또한 각 속성이 가지는 값의 종류와 범위를 명확하게 하기 위해 도메인정의를 미리 하여 정의하여 용어사전과 같이 사용한다. 용어사전과 도메인정의를 같이 사용하여 프로젝트를 진행할 경우 용어적 표준과 데이터타입의 일관

성을 확보할 수 있게 된다.
속성명을 부여하는 원칙은 (그림 1-1-29)와 같다.



1. 해당업무에서 사용하는 이름을 부여 한다.
2. 서술식 속성명은 사용하지 않는다.
3. 약어사용은 가급적 제한한다.
4. 전체 데이터모델에서 유일성 확보하는 것이 좋다.

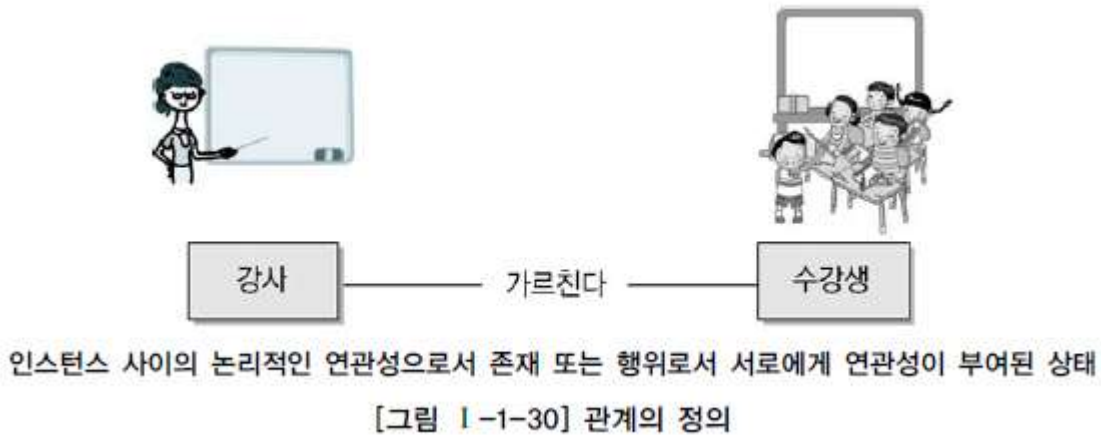
[그림 1-1-29] 속성의 명칭 부여

- 속성의 이름을 부여할 때는 현업에서 사용하는 이름을 부여하는 것이 가장 중요하다. 아무리 일반적인 용어라 할지라도 그 업무에서 사용되지 않으면 속성의 명칭으로 사용하지 않는 것이 좋다.
- 일반적으로는 서술식의 속성명은 사용하지 말아야 한다. 명사형을 이용하고 수식어가 많이 붙지 않도록 유의하여 작성한다. 수식어가 많으면 의미파악이 힘들고 상세 설계단계에서 물리속성으로 전환하는데 명확한 의미파악이 어렵게 된다. 소유격도 사용하지 않는다.
- 공용화되지 않은 업무에서 사용하지 않는 약어는 사용하지 않는 것이 좋다. 지나치게 약어를 많이 사용하면 업무분석자 내에서도 의사소통이 제약을 받으며 시스템을 운영할 때도 많은 불편을 초래할 수 있다.
- 가능하면 모든 속성의 이름은 유일하게 작성하는 것이 좋다. 물론 대량의 속성을 정의하는 경우 유일하게 작성하는 것이 어려울 수도 있지만 이렇게 하는 것이 나중에 데이터에 대한 흐름을 파악하고 데이터의 정합성을 유지하는데 큰 도움이 된다. 또한 반정규화(테이블통합, 분리, 칼럼의 중복 등)를 적용할 때 속성명의 충돌(Conflict)을 해결하여 안정적으로 반정규화를 적용할 수 있게 된다.

1. 관계의 개념

가. 관계의 정의

관계(Relationship)를 사전적으로 정의하면 상호 연관성이 있는 상태로 말할 수 있다. 이것을 데이터 모델에 대입하여 정의해 보면, “엔터티의 인스턴스 사이의 논리적인 연관성으로서 존재의 형태로 서나 행위로서 서로에게 연관성이 부여된 상태”라고 할 수 있다. 관계는 엔터티와 엔터티 간 연관성을 표현하기 때문에 엔터티의 정의에 따라 영향을 받기도 하고, 속성 정의 및 관계 정의에 따라서도 다양하게 변할 수 있다.

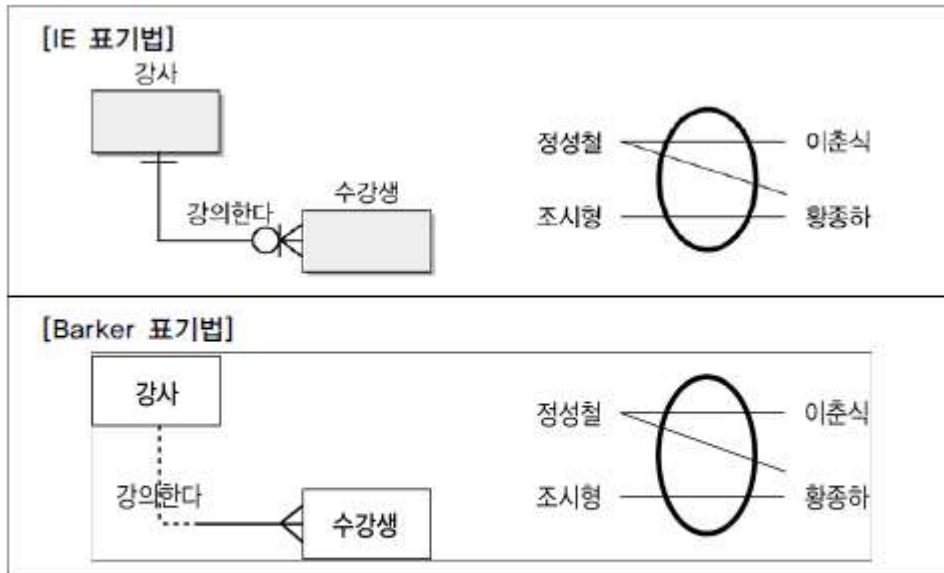


나. 관계의 패어링

유의해야할 점은 관계는 엔터티 안에 인스턴스가 개별적으로 관계를 가지는 것(패어링)이고 이것의 집합을 관계로 표현한다는 것이다. 따라서 개별 인스턴스가 각각 다른 종류의 관계를 가지고 있다면 두 엔터티 사이에 두 개 이상의 관계가 형성될 수 있다.

각각의 엔터티의 인스턴스들은 자신이 관련된 인스턴스들과 관계의 어커런스로 참여하는 형태를 관계 패어링(Relationship Paring)이라 한다. [그림 1-1-31]에서는 강사인 정성철은 이춘식과 황종하에게 강의를 하는 형태로 관계가 표현되어 있고 조시형은 황종하에게 강의를 하는 형태로 되어 있다. 이와 같이 엔터티내에 인스턴스와 인스턴스사이에 관계가 설정되어 있는 어커런스를 관계 패어링이라고 한다. 엔터티는 인스턴스의 집합을 논리적으로 표현하였다면 관계는 관계 패어링의 집합을 논리적으로 표현한 것이다.

최초의 ERD(Chen 모델)에서 관계는 속성을 가질 수 있었으나 요즘 ERD에서는 관계를 위해 속성을 도출하지는 않는다. 관계의 표현에는 이항 관계(Binary Relationship), 삼항 관계(Ternary Relationship), n 항 관계가 존재할 수 있는데 실제에 있어서 삼항 관계 이상은 잘 나타나지 않는다.

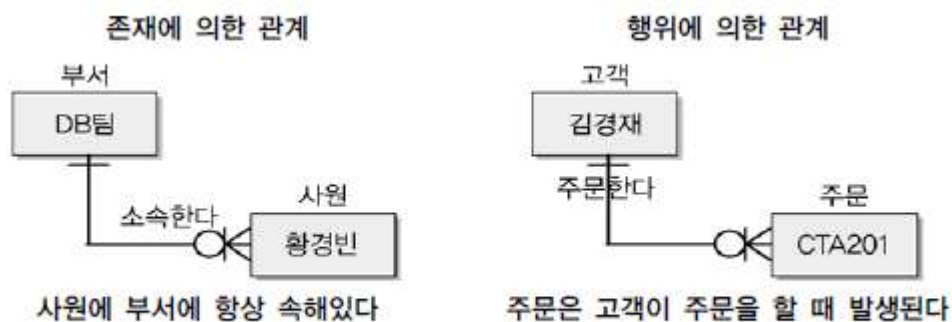


인스턴스 각각은 자신의 연관성을 가지고 있을 수 있음. 이것을 집합하여 '강의'라는 관계 도출

[그림 1-1-31] 관계의 패어링

2. 관계의 분류

관계가 존재에 의한 관계와 행위에 의한 관계로 구분될 수 있는 것은 관계를 연결함에 있어 어떤 목적으로 연결되었느냐에 따라 분류하기 때문이다.



[그림 1-1-32] 관계의 분류

[그림 1-1-32]에서 왼쪽 편에 있는 모델은 황경빈이란 사원이 DB 팀에 소속되어 있는 상태를 나타낸다. '소속된다'라는 의미는 행위에 따른 이벤트에 의해 발생하는 의미가 아니고 그냥 황경빈사원이 DB 팀에 소속되어 있기 때문에 나타나는 즉 존재의 형태에 의해 관계가 형성되어 있는 것이다

반면에 오른쪽에 있는 김경재 고객은 '주문한다'라는 행위를 하여 CTA201이라는 주문번호를 생성하였다. 주문 엔터티의 CTA201 주문번호는 김경재 고객이 '주문한다'라는 행위에 의해 발생되었기 때문에 두 엔터티 사이의 관계는 행위에 의한 관계가 되는 것이다.

UML(Unified Modeling Language)에는 클래스 다이어그램의 관계중 연관관계(Association)와 의존관계(Dependency)가 있다. 이 둘의 차이는 연관관계는 항상 이용하는 관계로 존재적 관계에 해당하고 의존관계는 상대방 클래스의 행위에 의해 관계가 형성될 때 구분하여 표현한다는 것이다. 즉, ERD에서는 존재적 관계와 행위에 의한 관계를 구분하지 않고 표현했다면 클래스 다이어그램에서는 이것을 구분하여 연관관계와 의존관계로 표현하고 있는 것이다. 연관관계는 표현방법이 실선으로

표현되고 소스코드에서 멤버변수로 선언하여 사용하게 하고, 의존관계는 점선으로 표현되고 행위를 나타내는 코드인 Operation(Method)에서 파라미터 등으로 이용할 수 있도록 되어 있다.

3. 관계의 표기법

관계에서는 표기법이 상당히 복잡하고 여러 가지 의미를 가지고 있다. 다음 3 가지 개념과 함께 표기법을 이해할 필요가 있다.

- 관계명(Membership) : 관계의 이름
- 관계차수(Cardinality) : 1:1, 1:M, M:N
- 관계선택사양(Optionality) : 필수관계, 선택관계

가. 관계명(Membership)

관계명은 엔터티가 관계에 참여하는 형태를 지칭한다. 각각의 관계는 두 개의 관계명을 가지고 있다. 또한 각각의 관계명에 의해 두 가지의 관점으로 표현될 수 있다.



[그림 1-1-33] 관계의 관계명

엔터티에서 관계가 시작되는 편을 관계시작점(The Beginning)이라고 부르고 받는 편을 관계끝점(The End)이라고 부른다. 관계 시작점과 끝점 모두 관계이름을 가져야 하며 참여자의 관점에 따라 관계이름이 능동적(Active)이거나 수동적(Passive)으로 명명된다. 관계명은 다음과 같은 명명규칙에 따라 작성해야 한다.

- 애매한 동사를 피한다. 예를 들면 ‘관계된다’, ‘관련이 있다’, ‘이다’, ‘한다’ 등은 구체적이지 않아 어떤 행위가 있는지 또는 두 참여자간 어떤 상태가 존재하는지 파악할 수 없다.
- 현재형으로 표현한다. 예를 들면 ‘수강을 신청했다’, ‘강의를 할 것이다’ 라는 식으로 표현해서는 안된다. ‘수강 신청한다’, ‘강의를 한다’ 로 표현해야 한다.

나. 관계차수(Degree/Cardinality)

두 개의 엔터티간 관계에서 참여자의 수를 표현하는 것을 관계차수(Cardinality)라고 한다. 가장 일반적인 관계차수 표현방법은 1:M, 1:1, M:N 이다. 가장 중요하게 고려해야 할 사항은 한 개의 관계가 존재하느냐 아니면 두 개 이상의 멤버십이 존재하는지를 파악하는 것이 중요하다.

관계차수를 표시하는 방법은 여러 가지 방법이 있지만 Crow’s Foot 모델에서는 선을 이용하여 표현한다. 한 개가 참여하는 경우는 실선을 그대로 유지하고 다수가 참여한 경우는(Many) 까마귀발과 같은 모양으로 그려준다.

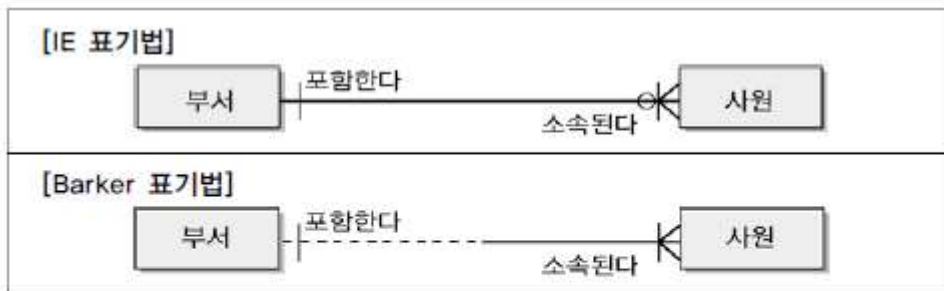
1) 1:1(ONE TO ONE) 관계를 표시하는 방법



[그림 1-1-34] 관계차수(1:1)

관계에 참여하는 각각의 엔터티는 관계를 맺는 다른 엔터티의 엔터티에 대해 단지 하나의 관계만을 가지고 있다.

2) 1:M(ONE TO MANY) 관계를 표시하는 방법

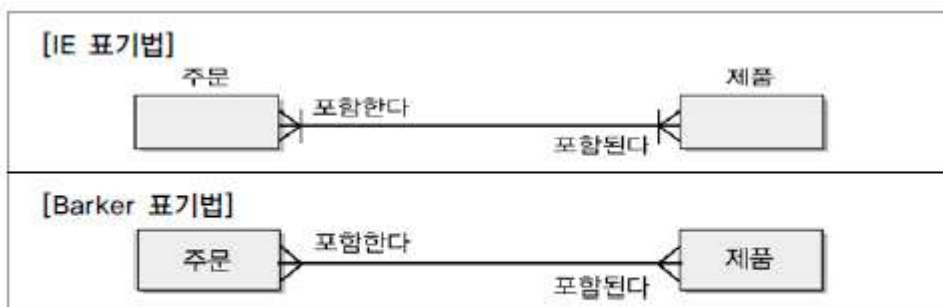


한 명의 직원은 한 부서에 소속되고 한 부서에는 여러 직원을 포함한다

[그림 1-1-35] 관계차수(1:M)

관계에 참여하는 각각의 엔터티는 관계를 맺는 다른 엔터티의 엔터티에 대해 하나나 그 이상의 수와 관계를 가지고 있다. 그러나 반대의 방향은 단지 하나만의 관계를 가지고 있다.

3) M:M(MANY TO MANY) 관계를 표시하는 방법



[그림 1-1-36] 관계차수(M:M)

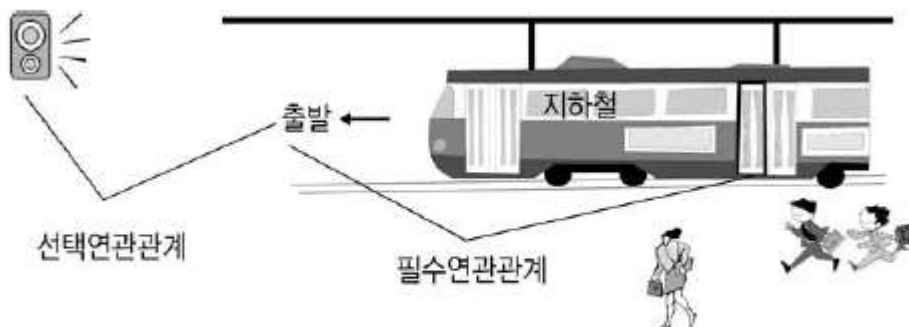
관계에 참여하는 각각의 엔터티는 관계를 맺는 다른 엔터티의 엔터티에 대해 하나나 그 이상의 수와 관계를 가지고 있다. 반대의 방향도 동일하게 관계에 참여하는 각각의 엔터티는 관계를 맺는 다른 엔터티의 엔터티에 대해 하나 또는 그 이상의 수와 관계를 가지고 있다. 이렇게 M:N 관계로 표현된 데이터 모델은 이후에 두 개의 주식별자를 상속받은 관계엔터티를 이용하여 3개의 엔터티로 구분하여 표현한다.

다. 관계선택사항(Optionality)

요즈음 웬만한 대도시에는 지하철이 많이 운행된다. 만약 지하철 문이 닫히지 않았는데 지하철이 떠난다면 무슨 일이 발생할까? 아마도 어떤 사람은 머리만 지하철 안에 들어오고 몸은 밖에 있는 채로 끌려갈 것이고, 또 어떤 사람은 가방만 지하철에 실어 보내는 사람도 있을 것이고, 지하철과 승강기 사이에 몸이 낄 수도 있을 것이다. 물론 지하철운행과 지하철문의 관계는 이렇게 설계되지 않

아 위와 같은 어처구니없는 일은 발생하지 않을 것이다. “반드시 지하철의 문이 닫혀야만 지하철은 출발한다.” 지하철출발과 지하철문닫힘은 필수(Mandatory)적으로 연결 관계가 있는 것이다. 이와 같은 것이 데이터 모델의 관계에서는 필수참여관계(Mandatory)가 된다.

또 지하철 안내방송시스템의 예를 들어보자. 지하철의 출발을 알리는 안내방송은 지하철의 출발과 상관없이 방송해도 아무런 문제가 발생하지 않는다. 물론 정해진 시간에 방송을 하면 승객에게 정보로서 유익하겠지만 꼭 그렇게 할 필요는 없다. 그래서 가끔씩 시스템의 녹음된 여성의 목소리가 아닌 시끄러운 남자 기사가 방송을 하는 경우가 있다. 안내방송시스템이 고장이 나도 지하철운행에는 별로 영향을 주지 않는다. 방송시점도 조금씩 다르게 나타나도 지하철이 출발하는 것과는 밀접하게 연관되지 않는다. 이와 같이 지하철의 출발과 지하철방송과는 정보로서 관련은 있지만 서로가 필수적인(Mandatory) 관계는 아닌 선택적인 관계(Optional)가 되는 것이다.



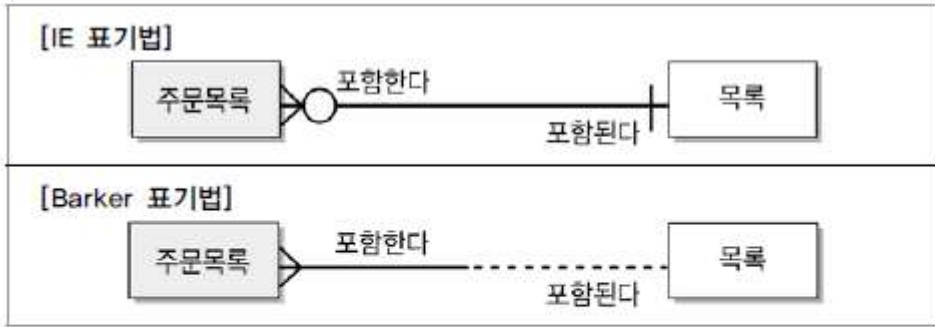
[그림 1-1-37] 관계선택사양

이와 같은 것이 데이터 모델 관계에서는 선택참여관계(Optional)가 된다. 참여하는 엔터티가 항상 참여하는지 아니면 참여할 수도 있는지를 나타내는 방법이 필수(Mandatory Membership)와 선택참여(Optional Membership)이다.

필수참여는 참여하는 모든 참여자가 반드시 관계를 가지는, 타 엔터티의 참여자와 연결이 되어야 하는 관계이다. 예를 들면 주문서는 반드시 주문목록을 가져야 하며 주문목록이 없는 주문서는 의미가 없으므로 주문서와 주문목록은 필수참여관계가 되는 것이다. 반대로 목록은 주문이 될 수도 있고 주문이 되지 않은 목록이 있을 수도 있으므로 목록과 주문목록과의 관계는 선택참여(Optional Membership)가 되는 것이다. 선택참여된 항목은 물리속성에서 Foreign Key 로 연결될 경우 Null 을 허용할 수 있는 항목이 된다.

만약 선택참여로 지정해야 할 관계를 필수참여로 잘못 지정하면 애플리케이션에서 데이터가 발생할 때 반드시 한 개의 트랜잭션으로 제어해야 하는 제약사항이 발생한다. 그러므로 설계단계에서 필수참여와 선택참여는 개발시점에 업무 로직과 직접적으로 관련된 부분이므로 반드시 고려되어야 한다. 선택참여관계는 ERD 에서 관계를 나타내는 선에서 선택참여하는 엔터티 쪽을 원으로 표시한다. 필수참여는 아무런 표시를 하지 않는다.

만약 관계가 표시된 양쪽 엔터티에 모두 선택참여가 표시된다면, 즉 0:0(Zero to Zero)의 관계가 된다면 그 관계는 잘못될 확률이 많으므로 관계설정이 잘못되었는지를 검토해 보아야 한다.



하나의 주문목록에는 한 개의 목록을 항상 포함하고
한 목록은 여러 개의 주문 목록에 의해 포함될 수 있다.

[그림 1-1-38] 관계선택참여

관계선택사양은 관계를 통한 상대방과의 업무적인 제약조건을 표현하는 것으로서 간단하면서 아주 중요한 표기법이다. 이것을 어떻게 설정했는지에 따라 참조무결성 제약조건이 바뀌게 되므로 주의 깊게 모델링을 해야 한다.

4. 관계의 정의 및 읽는 방법

가. 관계 체크사항

두 개의 엔터티 사이에서 관계를 정의할 때 다음 사항을 체크해 보도록 한다.

- 두 개의 엔터티 사이에 관심있는 연관규칙이 존재하는가?
- 두 개의 엔터티 사이에 정보의 조합이 발생하는가?
- 업무기술서, 장표에 관계연결에 대한 규칙이 서술되어 있는가?
- 업무기술서, 장표에 관계연결을 가능하게 하는 동사(Verb)가 있는가?

나. 관계 읽기

데이터 모델을 읽는 방법은 먼저 관계에 참여하는 기준 엔터티를 하나 또는 각(Each)으로 읽고 대상 엔터티의 개수(하나, 하나 이상)를 읽고 관계선택사양과 관계명을 읽도록 한다.

- 기준(Source) 엔터티를 한 개(One) 또는 각(Each)으로 읽는다.
- 대상(Target) 엔터티의 관계참여도 즉 개수(하나, 하나 이상)를 읽는다.
- 관계선택사양과 관계명을 읽는다.



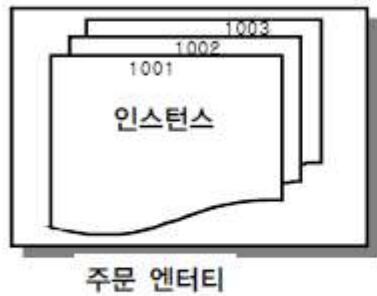
[그림 1-1-39] 관계의 읽는 방법

위의 관계를 정의를 한 사항에 대해서 뒷부분만 의문문으로 만들면 바로 관계를 도출하기 위한 질문 문장으로 만들 수 있다. 위의 질문을 업무를 분석하는 자기 스스로에게 질문하거나, 장표나 업무기술서 또는 업무를 잘 알고 있는 업무담당 고객과 대화를 하면서 관계를 완성해 갈 수 있다.

예를 들어, 주문과 제품과 관계를 질문하기 원할 때 “한 주문에 대해서 하나의 제품만을 주문합니까?”라고 할 수도 있고 또는 “한 제품은 하나의 주문에 대해서만 주문을 접수받을 수 있습니까?”라고 질문할 수 있다. 이러한 질문 방법은 엔터티간 관계설정뿐 아니라 업무의 흐름도 분석이 되는 실제 프로젝트에서 효과적인 방법이 된다.

1. 식별자(Identifiers) 개념

엔터티는 인스턴스들의 집합이라고 하였다. 여러 개의 집합체를 담고 있는 하나의 통에서 각각을 구분할 수 있는 논리적인 이름이 있어야 한다. 이 구분자를 식별자(Identifier)라고 한다. 식별자란 하나의 엔터티에 구성되어 있는 여러 개의 속성 중에 엔터티를 대표할 수 있는 속성을 의미하며 하나의 엔터티는 반드시 하나의 유일한 식별자가 존재해야 한다. 보통 식별자와 키(Key)를 동일하게 생각하는 경우가 있는데 식별자라는 용어는 업무적으로 구분이 되는 정보로 생각할 수 있으므로 논리 데이터 모델링 단계에서 사용하고 키는 데이터베이스 테이블에 접근을 위한 매개체로서 물리 데이터 모델링 단계에서 사용한다.



식별자는 엔터티내에서 인스턴스들을 구분할 수 있는 구분자이다.

[그림 1-1-40] 식별자의 정의

2. 식별자의 특징

주식별자인지 아니면 외부식별자인지 등에 따라 특성이 다소 차이가 있다. 먼저 주식별자일 경우 다음과 같은 특징을 갖는다.

- 주식별자에 의해 엔터티내에 모든 인스턴스들이 유일하게 구분되어야 한다.
- 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 한다.
- 지정된 주식별자의 값은 자주 변하지 않는 것이어야 한다.
- 주식별자가 지정이 되면 반드시 값이 들어와야 한다.

[표 1-1-7] 주식별자의 특징

특징	내용	비고
유일성	주식별자에 의해 엔터티내에 모든 인스턴스들을 유일하게 구분함	예) 사원번호가 주식별자가 모든 직원들에 대해 개인별로 고유하게 부여됨
최소성	주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 함	예) 사원번호만으로도 고유한 구조인데 사원분류코드+사원번호로 식별자가 구성될 경우 부절한 주식별자 구조임
불변성	주식별자가 한 번 특정 엔터티에 지정되면 그 식별자의 값은 변하지 않아야 함	예) 사원번호의 값이 변한다는 의미는 이전기록이 말소되고 새로운 기록이 발생하는 개념임
존재성	주식별자가 지정되면 반드시 데이터 값이 존재 (Null 안됨)	예) 사원번호 없는 회사직원은 있을 수 없음

대체식별자의 특징은 주식별자의 특징과 일치하지만 외부식별자는 별도의 특징을 가지고 있다. 외부식별자의 경우 주식별자 특징과 일치하지 않으며 참조무결성 제약조건(Referential Integrity)에 따른 특징을 가지고 있다.

3. 식별자 분류 및 표기법

가. 식별자 분류

식별자의 종류는 자신의 엔터티 내에서 대표성을 가지는가에 따라 주식별자(Primary Identifier)와 보조식별자(Alternate Identifier)로 구분하고 엔터티 내에서 스스로 생성되었는지 여부에 따라 내부식별자와 외부식별자(Foreign Identifier)로 구분할 수 있다. 또한 단일 속성으로 식별이 되는가에 따라 단일식별자(Single Identifier)와 복합식별자(Composit Identifier)로 구분할 수 있다. 원래 업무적으로 의미가 있던 식별자 속성을 대체하여 일련번호와 같이 새롭게 만든 식별자를 구분하기 위해 본질식별자와 인조식별자로도 구분할 수 있다.



[그림 1-1-41] 식별자의 분류

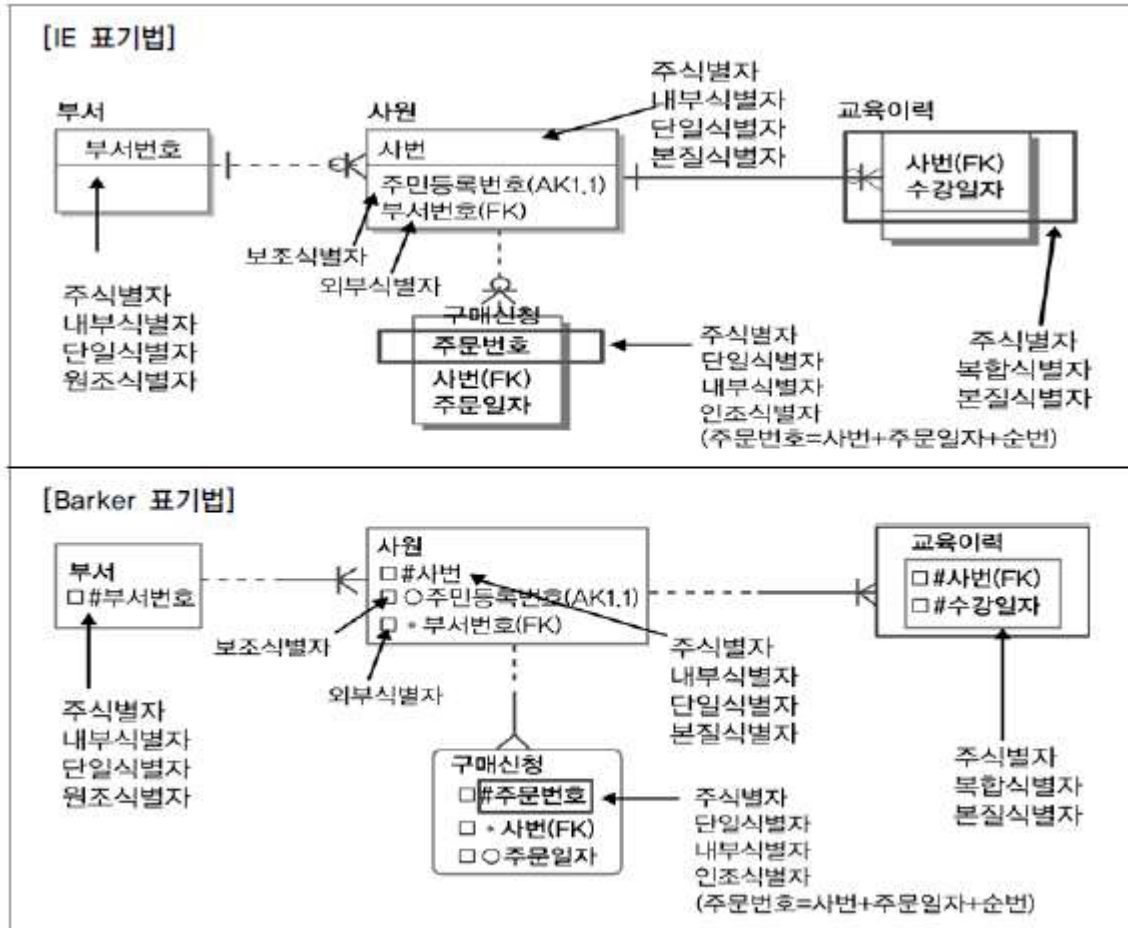
[그림 1-1-41]의 식별자에 대한 분류체계를 좀 더 상세하게 설명하면 [표 1-1-8]과 같이 표현할 수 있다.

[표 1-1-8] 식별자의 분류체계

분류	식별자	설명
대표성 여부	주식별자	엔터티 내에서 각 어커런스를 구분할 수 있는 구분자이며, 타 엔터티와 참조관계를 연결할 수 있는 식별자
	보조식별자	엔터티 내에서 각 어커런스를 구분할 수 있는 구분자이나 대표성을 가지지 못해 참조관계 연결을 못함
스스로 생성여부	내부식별자	엔터티 내부에서 스스로 만들어지는 식별자
	외부식별자	타 엔터티와의 관계를 통해 타 엔터티로부터 받아오는 식별자
속성의 수	단일식별자	하나의 속성으로 구성된 식별자
	복합식별자	둘 이상의 속성으로 구성된 식별자
대체 여부	본질식별자	업무에 의해 만들어지는 식별자
	인조식별자	업무적으로 만들어지지 않지만 원조식별자가 복잡한 구성을 가지고 있기 때문에 인위적으로 만든 식별자

나. 식별자 표기법

식별자에 대한 위의 분류방법을 데이터 모델에서 표현하면 [그림 1-1-42]와 같이 분류하여 설명할 수 있다.



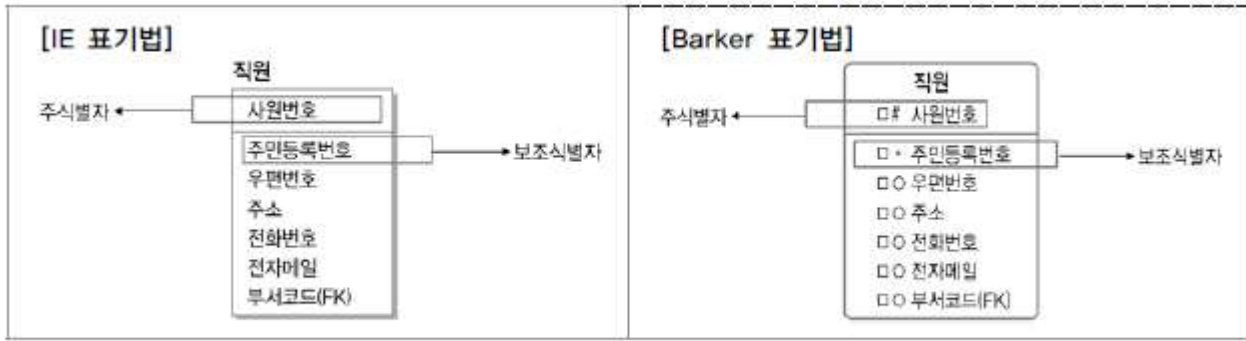
4. 주식별자 도출기준

데이터 모델링 작업에서 중요한 작업 중의 하나가 주식별자 도출작업이다. 주식별자를 도출하기 위한 기준을 정리하면 다음과 같다.

- 해당 업무에서 자주 이용되는 속성을 주식별자로 지정한다.
- 명칭, 내역 등과 같이 이름으로 기술되는 것들은 가능하면 주식별자로 지정하지 않는다.
- 복합으로 주식별자로 구성할 경우 너무 많은 속성이 포함되지 않도록 한다.

가. 해당 업무에서 자주 이용되는 속성을 주식별자로 지정하도록 함

예를 들면, 직원이라는 엔터티가 있을 때 유일하게 식별가능한 속성으로는 주민등록번호와 사원번호가 존재할 수 있다. 사원번호가 그 회사에서 직원을 관리할 때 흔히 사용되므로 사원번호를 주식별자로 지정하고 주민등록번호는 보조식별자로 사용할 수 있다.

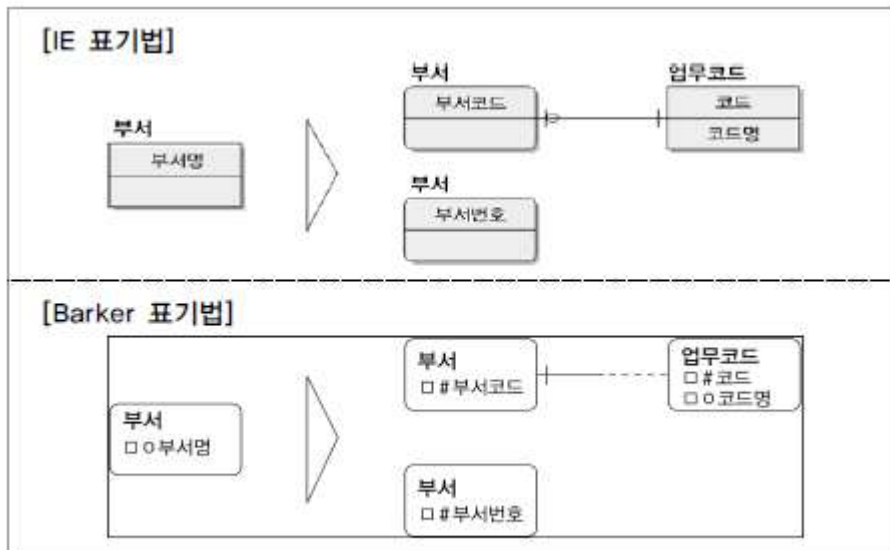


[그림 1-1-43] 식별자의 주식별자

나. 명칭, 내역 등과 같이 이름으로 기술되는 것은 피함

명칭, 내역 등과 같이 이름으로 기술되는 것들은 가능하면 주식별자로 지정하지 않도록 한다. 예를 들어, 한 회사에 부서이름이 100 개가 있다고 할 때, 각각의 부서이름은 유일하게 구별될 수 있다고 하여 부서이름을 주식별자로 지정하지 않도록 해야 한다. 만약 부서이름을 주식별자로 선정하면 물리데이터베이스로 테이블을 생성하여 데이터를 읽을 때 항상 부서이름이 WHERE 조건절에 기술되는 현상이 발생된다. 부서이름은 많은 경우 20 자 이상이 될 수 있으므로 조건절에 정확한 부서이름을 기술하기는 쉬운 일이 아니다.

이와 같이 명칭이나 내역이 있고 인스턴스들을 식별할 수 있는 다른 구분자가 존재하지 않을 경우는 새로운 식별자를 생성하도록 한다. 보통 일련번호와 코드를 많이 사용한다.



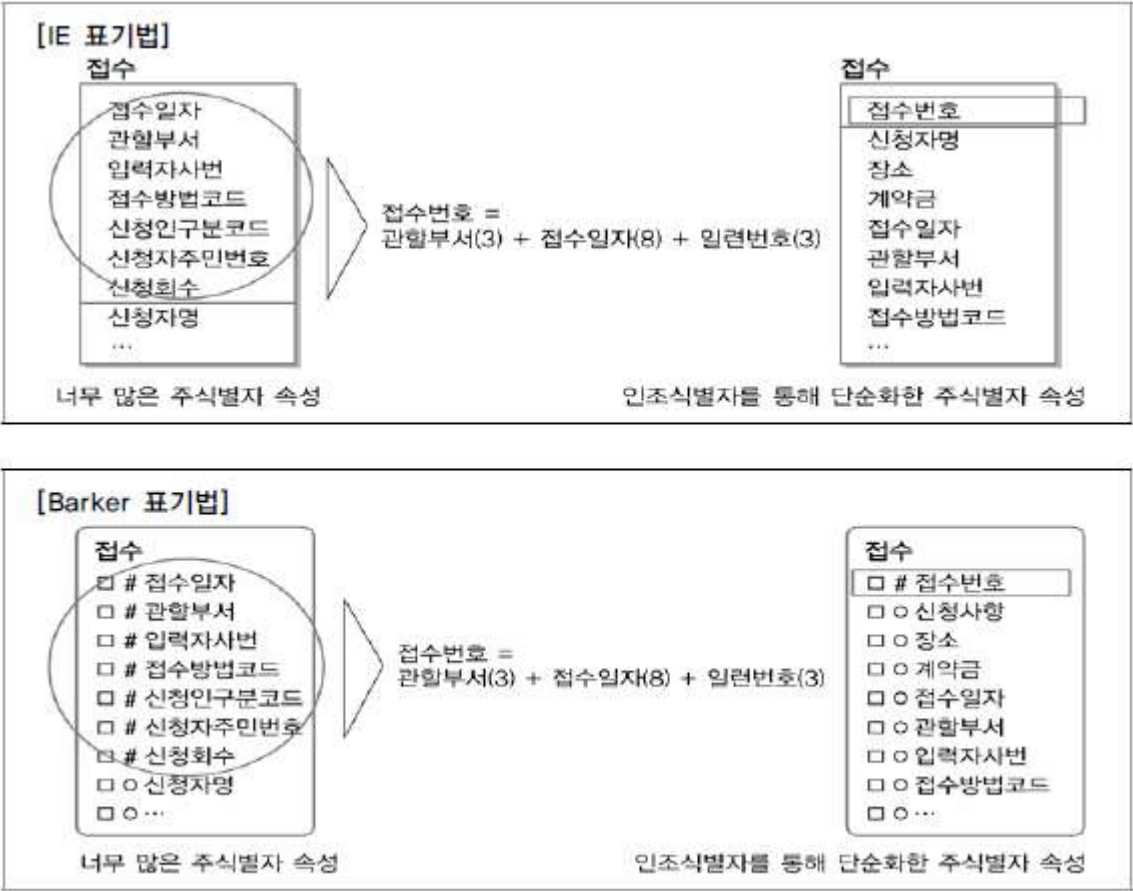
[그림 1-1-44] 주식별자-명칭/내역

부서명과 같은 경우는 부서코드를 부여하여 코드엔티에 등록한 후 부서코드로 주식별자를 지정하는 방법과 부서일련번호(부서번호)를 주식별자로 하고 부서명은 보조식별자로 활용하는 두 가지 방법이 있다.

다. 속성의 수가 많아지지 않도록 함

주식별자로 선정하기 위한 속성이 복합으로 구성되어 주식별자가 될 수 있을 때 가능하면 주식별자 선정하기 위한 속성의 수가 많지 않도록 해야 한다. 그러나 만약 주식별자로 선정된 속성들이 자신이 가지고 있는 자식엔티로부터 손자엔티, 그리고 증손자엔티까지 계속해서 상속이 되는 속성이고 복잡한 데이터 모델이 구현되어 물리데이터베이스에서 조인으로 인한 성능저하가 예상되는 모습을 가지고 있다면 속성의 반정규화 측면에서 하나의 테이블에 많은 속성이 있는 것이 인정될 수

도 있다. 하지만 일반적으로 주식별자의 속성의 개수가 많다는 것(일반적으로 7~8 개 이상)은 새로운 인조식별자(Artificial Identifier)를 생성하여 데이터 모델을 구성하는 것이 데이터 모델을 한층 더 단순하게 하고 애플리케이션을 개발할 때 조건절을 단순하게 할 수 있는 방법이 될 수 있다.



[그림 1 -1-45] 주식별자-복합속성

[그림 1 -1-45]는 왼편의 복잡한 주식별자를 임의의 주식별자를 생성하여 단순화한 예를 보여준다. 왼편 그림에서는 접수라는 엔터티에 어떤 부류의 사람이 특정접수방법에 의해 특정 날짜에 여러 번 신청하는 것을 관할부서에서 접수담당자가 입력한 대로 데이터가 식별될 수 있는 업무 규칙이 있는 경우이다. 접수의 주식별자가 접수일자, 관할부서, 입력자사번, 접수방법코드, 신청인구분코드, 신청자주민번호, 신청회수 등 7 개 이상의 복잡한 속성을 가지고 있다. 이러한 모델의 경우 실제 테이블에 Primary Key 는 7 개가 생성될 것이고 만약 특정 신청인의 계약금 하나만 가져온다고 하더라도 다음과 같이 복잡한 SQL 문장을 구사해야 한다.

```
SELECT 계약금 FROM 접수 WHERE 접수.접수일자 = '2010.07.15' AND 접수.관할부서 = '1001' AND 접수.입력자사번 = 'AB45588' AND 접수.접수방법코드 = 'E' AND 접수.신청인구분코드 = '01' AND 접수.신청인주민번호 = '7007171234567' AND 접수.신청횟수 = '1'
```

이렇게 된 SQL 문장을 접수번호라고 하는 인조식별자로 대체했다고 하면 특정신청인의 계약금 조회는 다음과 같이 간단하게 할 수 있다.

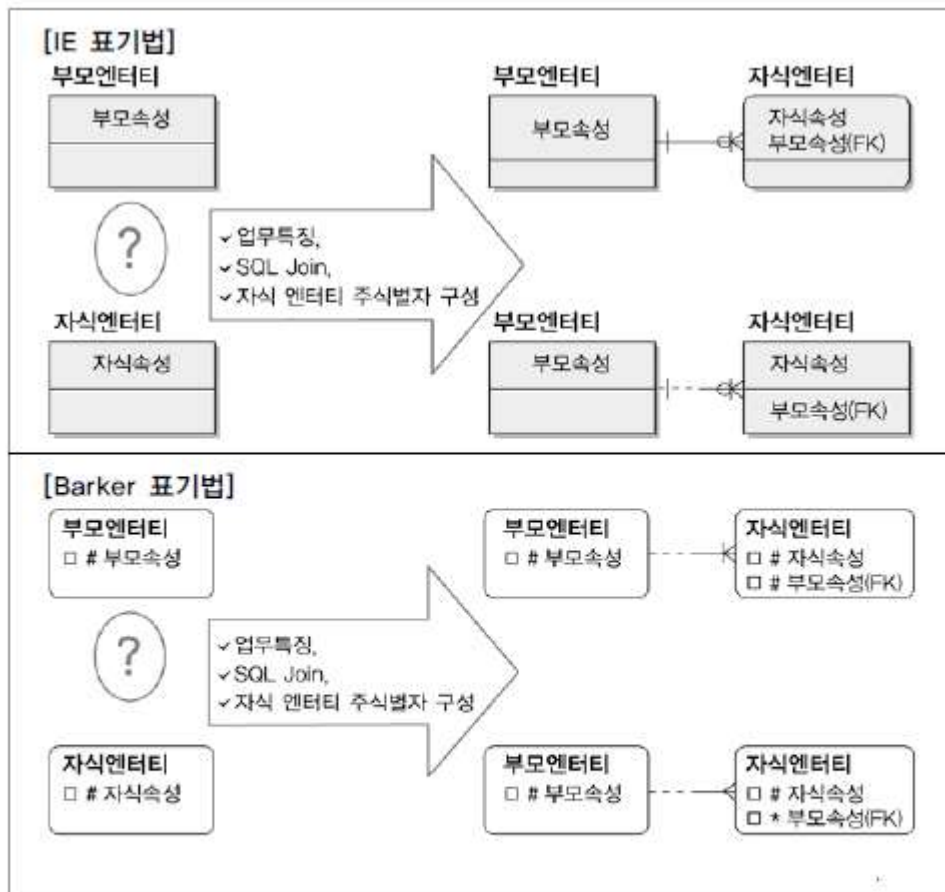
```
SELECT 계약금 FROM 접수 WHERE 접수.접수일자 = '100120100715001'
```

만약 접수 엔터티가 자식과 손자엔터티를 가지고 있고 자식과 손자엔터티에 있는 데이터를 서로 조인하여 가져오고자 한다면 아무리 간단한 SQL 문장이라도 쉽게 A4 용지 한 페이지는 넘어갈 것이다. 이렇게 모델상에 표현하는 문장의 간편성뿐만 아니라 애플리케이션 구성에 있어서도 복잡한 소스구성을 피하기 위하여 과도한 복합키는 배제하도록 노력해야 한다.

5. 식별자관계와 비식별자관계에 따른 식별자

가. 식별자관계와 비식별자 관계의 결정

외부식별자(Foreign Identifier)는 자기 자신의 엔터티에서 필요한 속성이 아니라 다른 엔터티와의 관계를 통해 자식 쪽에 엔터티에 생성되는 속성을 외부식별자라 하며 데이터베이스 생성 시에 Foreign Key 역할을 한다. 관계와 속성을 정의하고 주식별자를 정의하면 논리적인 관계에 의해 자연스럽게 외부식별자가 도출되지만 중요하게 고려해야 할 사항이 있다. 엔터티에 주식별자가 지정되고 엔터티간 관계를 연결하면 부모쪽의 주식별자를 자식엔터티의 속성으로 내려 보낸다. 이 때 자식엔터티에서 부모엔터티로부터 받은 외부식별자를 자신의 주식별자로 이용할 것인지 또는 부모와 연결이 되는 속성으로서만 이용할 것인지를 결정해야 한다.



엔터티 사이 관계유형은 업무특징, 자식엔터티의 주식별자구성, SQL 전략에 의해 결정된다.

[그림 1-1-46] 식별자/비식별자관계 조정

나. 식별자관계

부모로부터 받은 식별자를 자식엔터티의 주식별자로 이용하는 경우는 Null 값이 오면 안되므로 반드시 부모엔터티가 생성되어야 자기 자신의 엔터티가 생성되는 경우이다. 부모로부터 받은 속성을 자식엔터티가 모두 사용하고 그것만으로 주식별자로 사용한다면 부모엔터티와 자식엔터티의 관계는 1:1의 관계가 될 것이고 만약 부모로부터 받은 속성을 포함하여 다른 부모엔터티에서 받은 속성을 포함하거나 스스로 가지고 있는 속성과 함께 주식별자로 구성되는 경우는 1:M 관계가 된다.



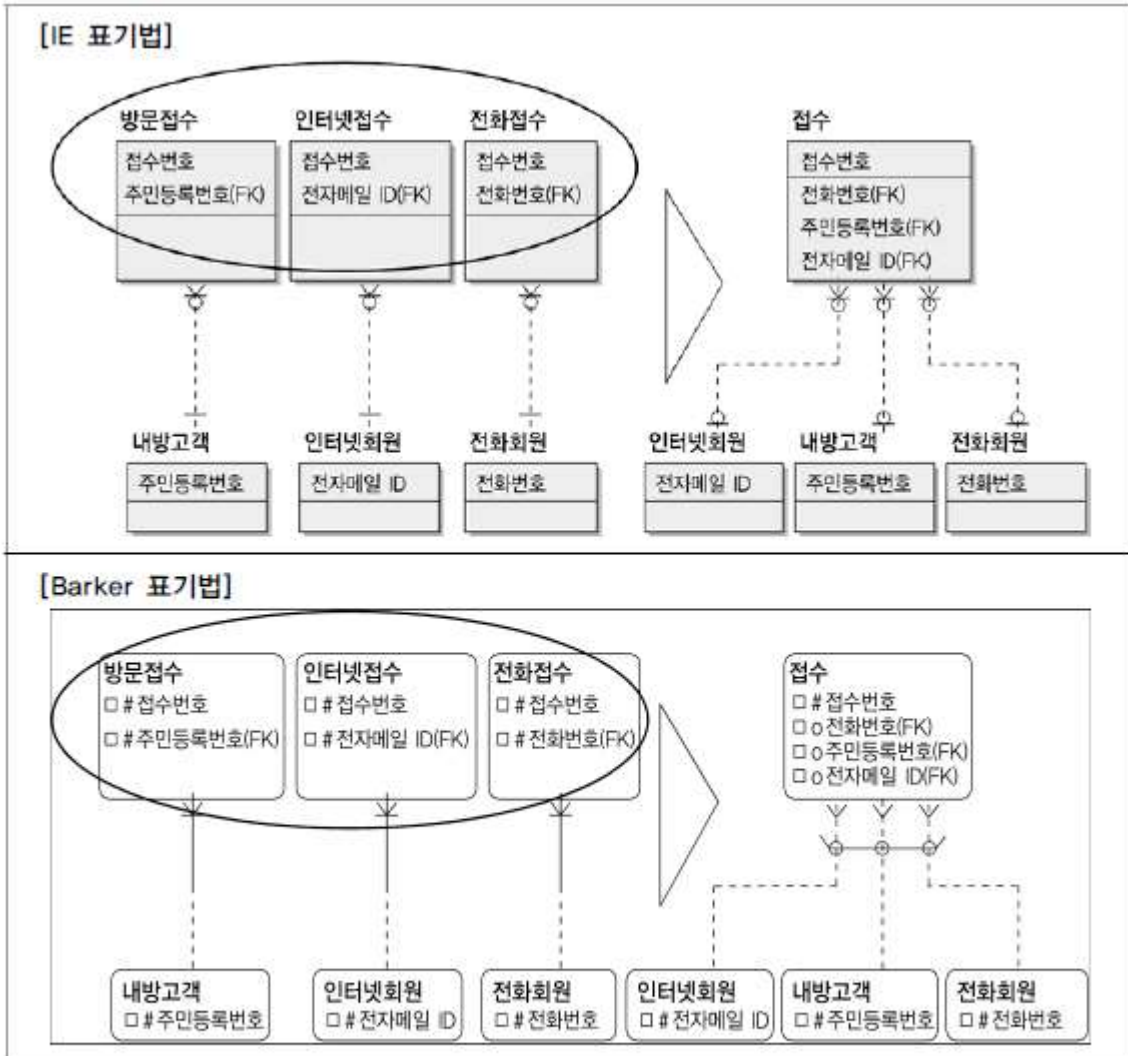
[그림 1-1-47] 외부식별자의 주식별자 역할

[그림 1-1-47]에서 발령엔터티는 반드시 사원엔터티가 있어야 자신도 생성될 수 있고 자신의 주식별자도 부모엔터티의 외부식별자 사원번호와 자신의 속성 발령번호로 이루어져 있음을 알 수 있다. 이 때 사원과 발령의 관계는 1:M 관계이다. 또한 사원과 임시직사원의 관계와 주식별자를 보면, 임시직사원의 주식별자는 사원의 주식별자와 동일하게 이용되는 경우를 볼 수 있다. 1:1 관계에서 이와 같이 나타나며 주식별자가 동일하며 엔터티 통합의 대상이 됨을 알 수 있다. 이와 같이 자식엔터티의 주식별자로 부모의 주식별자가 상속이 되는 경우를 식별자 관계(Identifying Relationship)라고 지칭한다.

다. 비식별자관계

부모엔터티로부터 속성을 받았지만 자식엔터티의 주식별자로 사용하지 않고 일반적인 속성으로만 사용하는 경우가 있다. 이와 같은 경우를 비식별자 관계(Non-Identifying Relationship)라고 하며 다음의 네 가지 경우에 비식별자 관계에 의한 외부속성을 생성한다.

- 1) 자식엔터티에서 받은 속성이 반드시 필수가 아니어도 무방하기 때문에 부모 없는 자식이 생성될 수 있는 경우이다.
- 2) 엔터티별로 데이터의 생명주기(Life Cycle)를 다르게 관리할 경우이다. 예를 들어 부모엔터티에 인스턴스가 자식의 엔터티와 관계를 가지고 있었지만 자식만 남겨두고 먼저 소멸될 수 있는 경우가 이에 해당된다. 이에 대한 방안으로 물리데이터베이스 생성 시 Foreign Key 를 연결하지 않는 임시적인 방법을 사용하기도 하지만 데이터 모델상에서 관계를 비식별자관계로 조정하는 것이 가장 좋은 방법이다.
- 3) 여러 개의 엔터티가 하나의 엔터티로 통합되어 표현되었는데 각각의 엔터티가 별도의 관계를 가질 때이며 이에 해당된다.



엔티티가 통합되면서 원래 식별자관계가 비식별자관계로 될 수 밖에 없게 됨

[그림 1-1-48] 외부식별자의 비식별자 역할

[그림 1-1-48]은 접수엔티티가 인터넷접수, 내방접수, 전화접수가 하나로 통합되어 표현되어 있는 경우에 해당되며 통합된 엔티티에 각각의 엔티티가 별도의 관계를 가지고 있고 각각의 관계로부터 받은 주식별자를 접수엔티티의 주식별자로 사용할 수 없는 모습을 보여준다.

4) 자식엔티티에 주식별자로 사용하여도 되지만 자식엔티티에서 별도의 주식별자를 생성하는 것이 더 유리하다고 판단될 때 비식별자 관계에 의한 외부식별자로 표현한다.

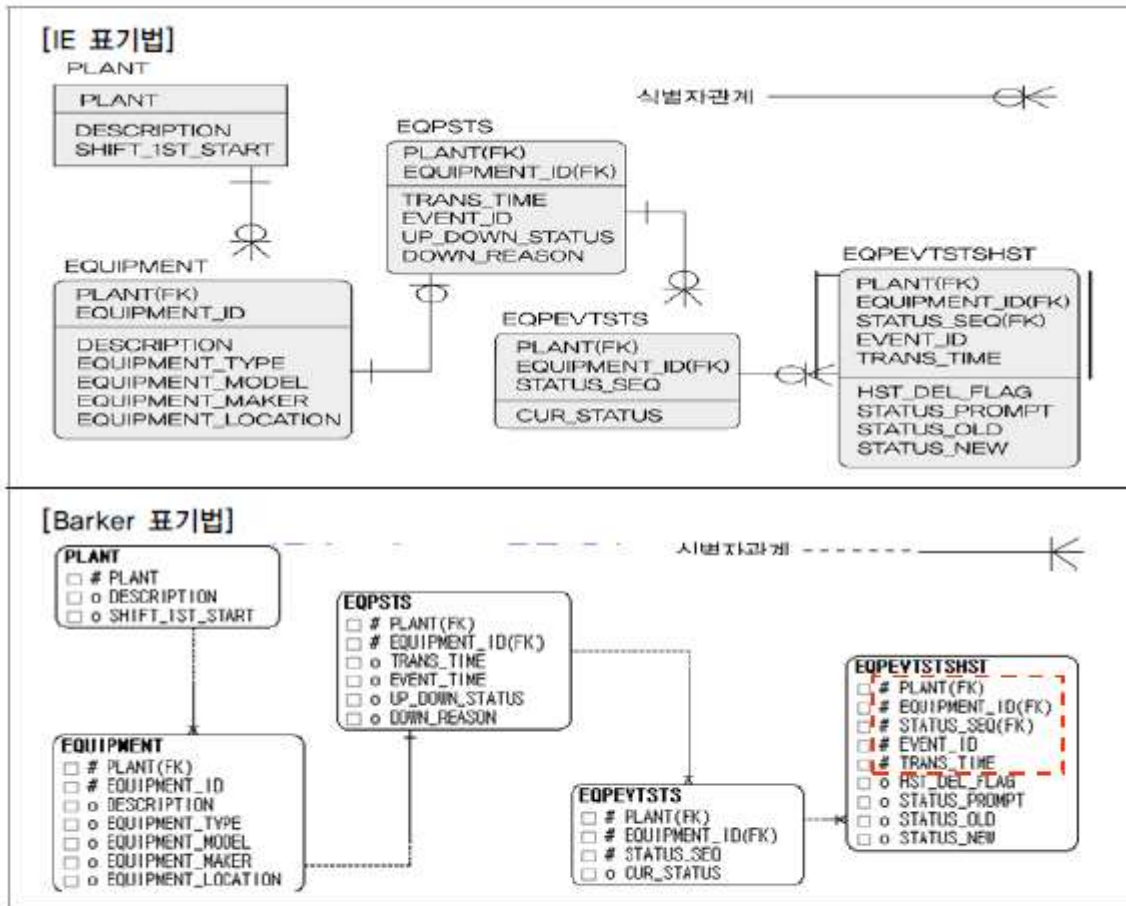


[그림 1-1-49] 자식엔티티의 독립 주식별자

[그림 1-1-49]는 계약이 반드시 직원에 의해 이루어져 직원번호와 계약번호로 주식별자를 구성할 수 있지만 계약번호 단독으로도 계약 엔티티의 주식별자를 구성할 수 있으므로 하나만 가지고 있는 것이 더 효율적이라고 판단하여 계약번호만 주식별자로 하고 계약직원번호는 일반속성 외부식별자로서 사용하게 된 경우이다.

라. 식별자 관계로만 설정할 경우의 문제점

단지 식별자관계와 비식별자관계에 대한 설정을 고려하지 않은 것이 개발의 복잡성을 증가시키는 요인이 될까?



부모에서 자식으로 식별자 관계로 연결되므로 인해 주식별자의 속성 수가 많아지게 된다.

[그림 1-1-50] 식별자관계 연결의 주식별자

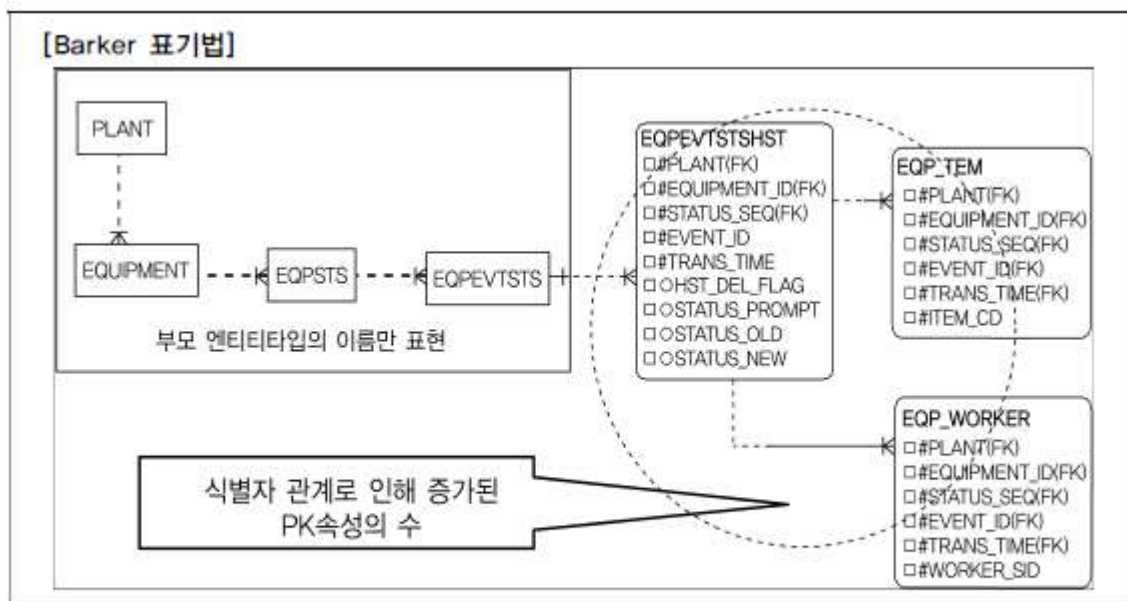
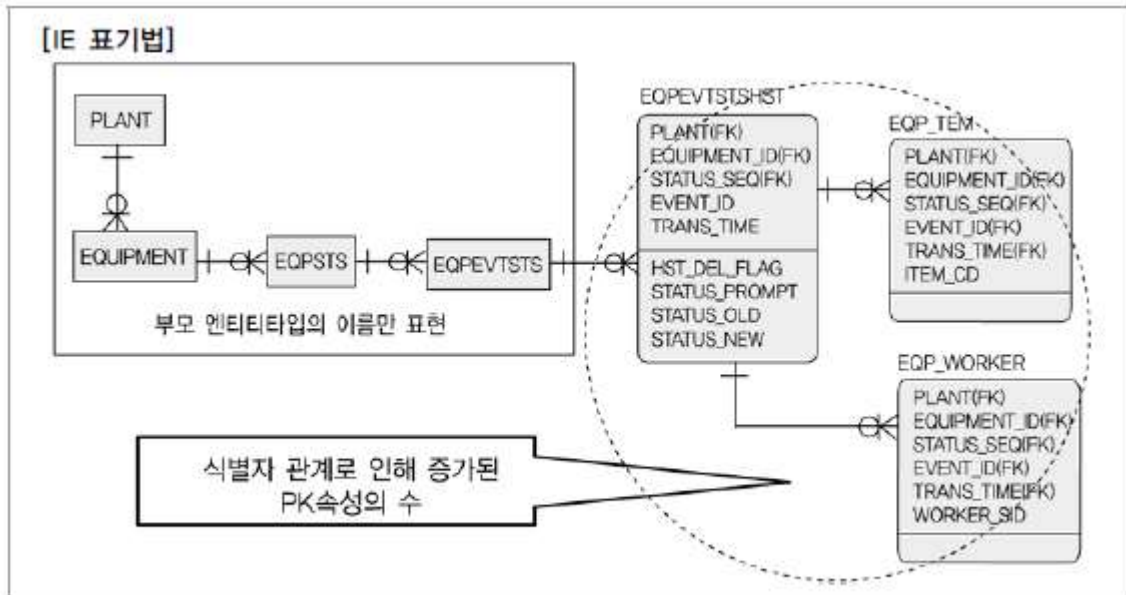
[그림 1-1-50]의 데이터 모델은 원시 엔터티였던 PLANT의 경우 단지 한 개의 속성만이 PK 속성이었으나 EQPEVTSTSHST 엔터티의 경우 부모로부터 모두 식별자관계 연결로 인해 PK 속성의 개수가 무려 5개나 설정될 수 밖에 없게 된 사례이다.

즉, PLANT 엔터티에는 PK 속성의 수가 1개이고 관계가 1:M으로 전개되었으므로 자식엔터티는 PLANT 엔터티의 PK 속성의 수 + 1이 성립된다. 물론 1개 이상의 속성의 추가되어야 1:M 관계를 만족할 수 있다. 이와 같은 원리에 의해 1:M 관계의 식별자관계의 PK 속성의 수는 다음과 같다.

원 부모엔터티 : 1개 2대 부모엔터티 : 2개 이상 = 원부모 1개 + 추가 1개 이상 + 3대 부모엔터티 : 3개 이상 = 원부모 1개 + 2대 1개 + 추가 1개 이상 3대 부모엔터티 : 3개 이상 = 원부모 1개 + 2대 1개 + 3대 1개 + 추가 1개 이상 4대 부모엔터티 : 4개 이상 = 원부모 1개 + 2대 1개 + 3대 1개 + 4대 1개 + 추가 1개 이상

이와 같은 규칙에 의해 지속적으로 식별자 관계를 연결한 데이터 모델의 PK 속성의 수는 데이터 모델의 흐름이 길어질수록 증가할 수 밖에 없는 구조를 가지게 된다.

[그림 1-1-50]의 예시 모델에서 EQPEVTSTSHST에 설정된 PK 속성을 이용하여 해당 이력의 수리 ITEM과 작업자에 대한 엔터티가 별도로 발생이 가능한 모델은 다음 [그림 1-1-51]과 같이 PK 속성 수가 많은 데이터 모델을 만들 수 밖에 없다.



세 개의 테이블에서 정보를 가져오는 SQL 구문을 만들면 어떻게 될까?

[그림 1-1-51] 식별자관계 연결의 주식별자

예시 모델의 맨 하위에 있는 EQPEVTSTSHST 에서 다시 새로운 엔터티 EQP_ITEM, EQP_WORKER 와 관계를 맺고 있을 때 이 세 개의 엔터티에서 정보를 가져오는 SQL 구문을 작성해 보자.

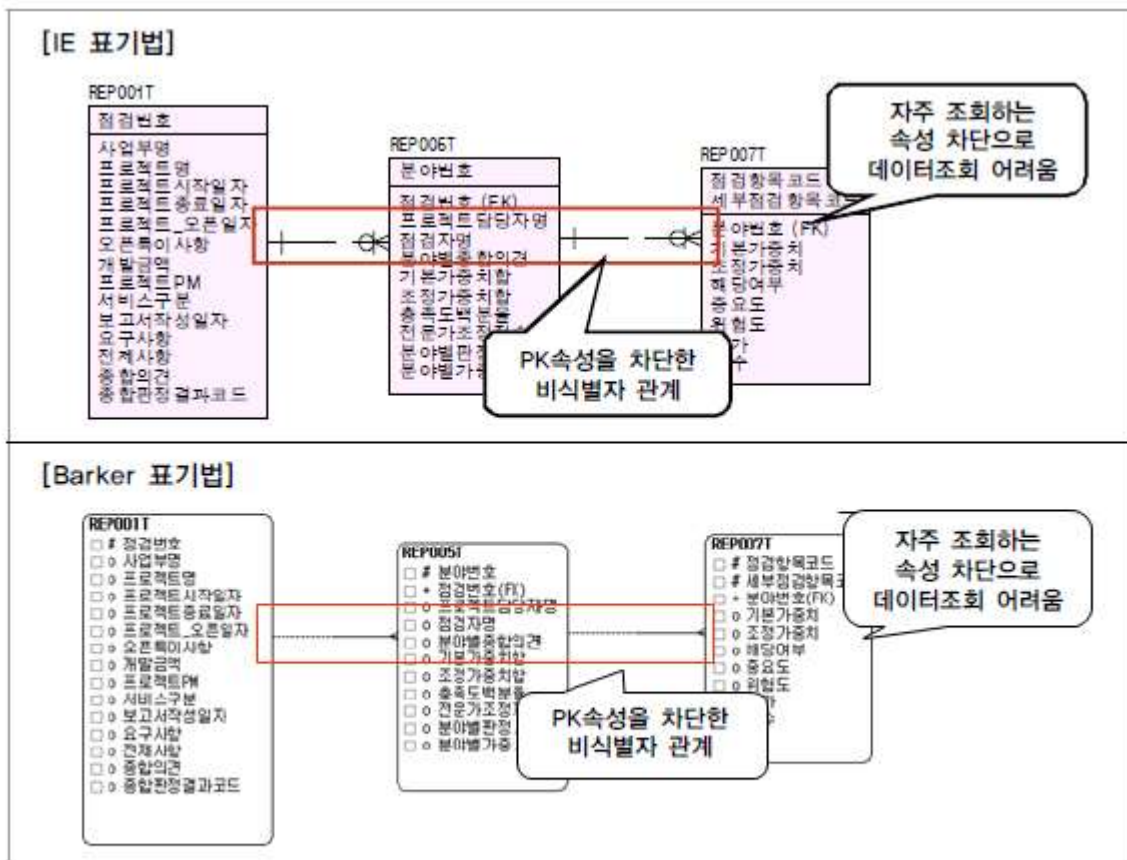
```
SELECT A.EVENT_ID, A.TRANS_TIME, A.HST_DEL_FLAG, A.STATUS_PROMPT, A.STATUS_OLD,
A.STATUS_NEW FROM EQPEVTSTSHST A, EQP_ITEM B, EQP_WORKER C WHERE A.PLANT =
B.PLANT AND A.EQUIPMENT_ID = B.EQUIPMENT_ID AND A.STATUS_SEQ = B.STATUS_SEQ AND
A.EVENT_ID = B.EVENT_ID AND A.TRANS_TIME = B.TRANS_TIME AND B.ITEM_CD = 'A001' AND
A.PLANT = C.PLANT AND A.EQUIPMENT_ID = C.EQUIPMENT_ID AND A.STATUS_SEQ =
C.STATUS_SEQ AND A.EVENT_ID = C.EVENT_ID AND A.TRANS_TIME = C.TRANS_TIME AND
C.WORKER_SID = 'A012008001'
```

고작 3 개 정도의 엔터티를 조인했을 뿐인데 SQL 구문의 WHERE 절이 매우 길어진 사실을 확인할 수 있다. 실제로 프로젝트에서는 개발자가 개발할 때 당연히 데이터 모델을 참조하면서 엔터티와 관계를 이용하여 개발해야 하는데 생성된 엔터티 스키마 정보만을 보고 개발하는 경우가 많다. 그런데 위와 같이 조인에 참여하는 주식별자속성의 수가 많을 경우 정확하게 조인관계를 설정하지 않

고 즉, 누락하여 개발하는 경우가 간혹 발견되기도 한다. 정리하면 식별자 관계만으로 연결된 데이터 모델의 특징은 주식별자 속성이 지속적으로 증가할 수 밖에 없는 구조로서 개발자 복잡성과 오류가능성을 유발시킬 수 있는 요인이 될 수 있다는 사실을 기억해야 한다.

마. 비식별자 관계로만 설정할 경우의 문제점

일반적으로 각각의 엔터티에는 중요한 기준 속성이 있는데 이러한 기준속성은 부모엔터티에 있는 PK 속성으로부터 상속되어 자식엔터티에 존재하는 경우가 많다. 이러한 속성의 예로 ‘주민등록번호’, ‘사원번호’, ‘주문번호’, ‘목록번호’ 등이 있다. 이런 속성은 부모엔터티를 조회할 때도 당연히 쓰이지만 자식엔터티의 데이터를 조회할 때도 해당 조건이 조회의 조건으로 걸리는 경우가 다수이다. 그런데 데이터 모델링을 전개할 때 각 엔터티 간의 관계를 비식별자 관계로 설정하면 이런 유형의 속성이 자식엔터티로 상속이 되지 않아 자식엔터티에서 데이터를 처리할 때 쓸데없이 부모엔터티까지 찾아가야 하는 경우가 발생된다.







테이블 REP007T에서 점검번호='301'인 데이터를 조회해 보자

[그림 1-1-52] 비식별자관계

[그림 1-1-52]의 예에서는 REP001T, REP005T, REP007T 간의 관계가 비식별자 관계로 연결되면서 점검번호, 분야번호 속성이 관계를 타고 자식엔터티로 내려가는 것을 차단하였다. 이러한 모델에서는 만약 위 모델에 있는 맨 하위에 있는 REP007T 엔터티에서 어떤 점검에 대한 정보를 보려고 하면 불필요한 조인이 다량으로 유발되면서 SQL 구문도 길어지고 성능이 저하되는 현상이 발생이 된다.

[표 1-1-9] 식별자와 비식별자관계 비교

비식별자관계	식별자 관계
<p>[IE 표기법]</p>  <p>[Barker 표기법]</p>  <p>SELECT A, 기본가중치, A, 조정가중치 FROM REP007T A, REP005T, REP001T C WHERE A, 분야번호 = B, 분야번호 AND B, 점검번호 = C, 점검번호 AND C, 점검번호 = '105'</p>	<p>[IE 표기법]</p>  <p>[Barker 표기법]</p>  <p>SELECT A, 기본가중치, A, 조정가중치 FROM REP007T A, REP005T, REP001T C WHERE A, 점검번호 = '105'</p>
테이블 REP007T에서 점검번호='301'인 데이터를 조회	

[표 1-1-9]의 조회 패턴을 보면 고작 점검번호='301' 정도로서 간단하면서도 이 업무에서는 가장 근간이 되는 조회의 패턴 정보로 여겨지는 조건이다. 이 조건은 아마도 웬만한 업무처리에는 많이 포함될 것으로 보이는데 단순하게 걸리는 이 하나의 조회 조건도 왼쪽과 같이 비식별자 관계로만 데이터 모델링을 전개하다 보면 SQL 구문에 많은 조인이 걸리게 되고 그에 따라 복잡성이 증가하고 성능이 저하되게 되는 것이다.

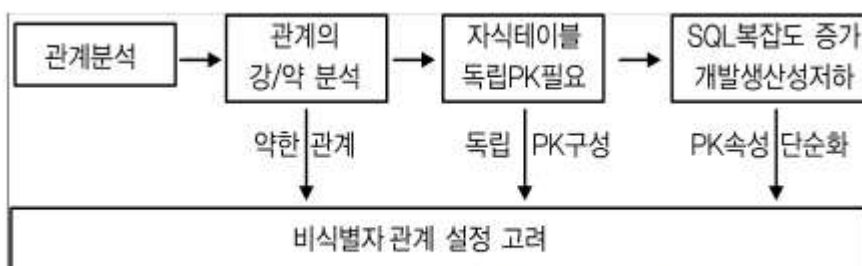
오른쪽과 같이 식별자관계를 통해 연결하다보면, 부모의 모든 주식별자 속성을 상속받음으로 인해 맨 하위에 있는 자식엔터티에서 바로 조회의 조건을 이용하여 원하는 정보를 가져올 수 있다. 이러한 경우는 당연히 성능과 개발의 용이성 측면에서는 식별자관계가 우위에 있음을 보여준다.

따라서 이 두 가지 경우에 대해서 일정한 규칙을 가지고 데이터 모델링을 하는 기술이 필요하며, 다음에 제시된 고려사항을 데이터 모델링에 반영한다면 효과적인 데이터 모델을 만들어 내는데 유용하게 활용할 수 있다.

바. 식별자관계와 비식별자관계 모델링

1) 비식별자관계 선택 프로세스

실제로 프로젝트를 전개할 때 식별자관계와 비식별자관계를 취사선택하여 연결하는 내공은 높은 수준의 기술을 요하고 있다. 특히 식별자관계에서 비식별자관계를 파악하는 기술이 필요한데 다음 흐름(Flow)에 따라 비식별자관계를 선정한다면 합리적으로 관계를 설정하는 모습이 될 수 있다. 기본적으로 식별자관계로 모든 관계가 연결되면서 다음 조건에 해당할 경우 비식별자관계로 조정하면 된다.



[그림 1-1-53] 비식별자관계 설정 고려사항

여기에서 가장 중요한 요인은 자식엔터티의 독립된 주식별자 구성이 필요한지를 분석하는 부분이다. 독립적으로 주식별자를 구성한다는 의미는 업무적 필요성과 성능상 필요여부를 모두 포함하는 의미로 이해하면 된다.

2) 식별자와 비식별자관계 비교

강한 관계인 식별자관계와 약한 관계인 비식별자관계를 비교하면 [표 1-1-10]과 같이 나타낼 수 있다.

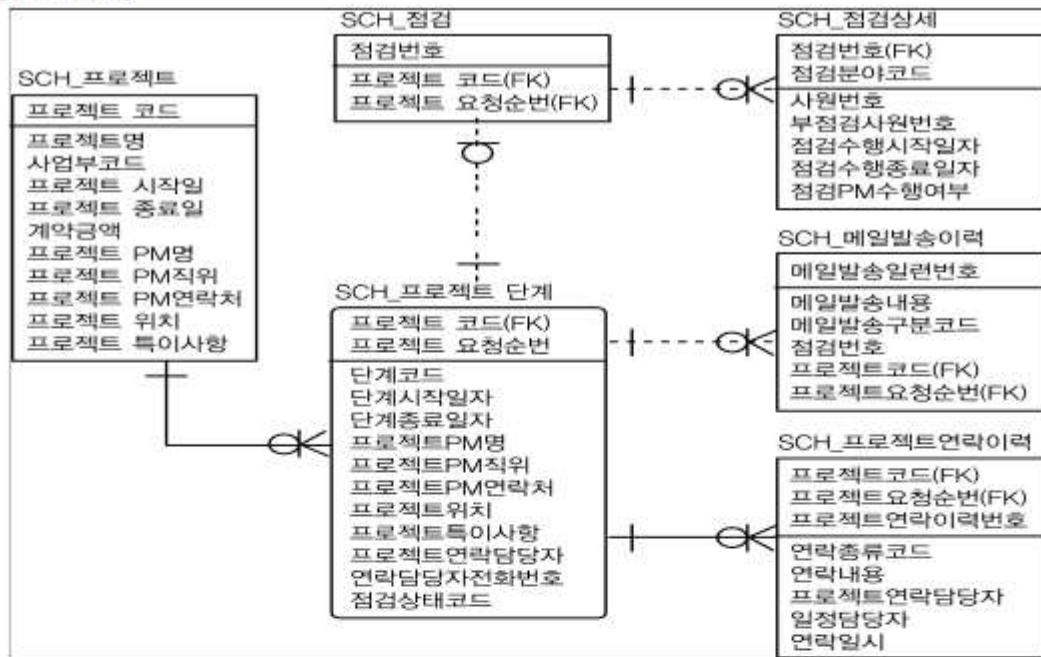
[표 1-1-10] 식별자와 비식별자관계 비교

항목	식별자관계	비식별자관계
목적	강한 연결관계 표현	약한 연결관계 표현
자식 주식별자 영향	자식 주식별자의 구성에 포함됨	자식 일반 속성에 포함됨
표기법	실선 표현	점선 표현
연결 고려사항	<ul style="list-style-type: none"> - 반드시 부모엔터티 종속 - 자식 주식별자구성에 부모 주식별자포함 필요 - 상속받은 주식별자속성을 타 엔터티에 이전 필요 	<ul style="list-style-type: none"> - 약한 종속관계 - 자식 주식별자구성을 독립적으로 구성 - 자식 주식별자구성에 부모 주식별자 부분 필요 - 상속받은 주식별자속성을 타 엔터티에 차단 필요 - 부모쪽의 관계참여가 선택관계

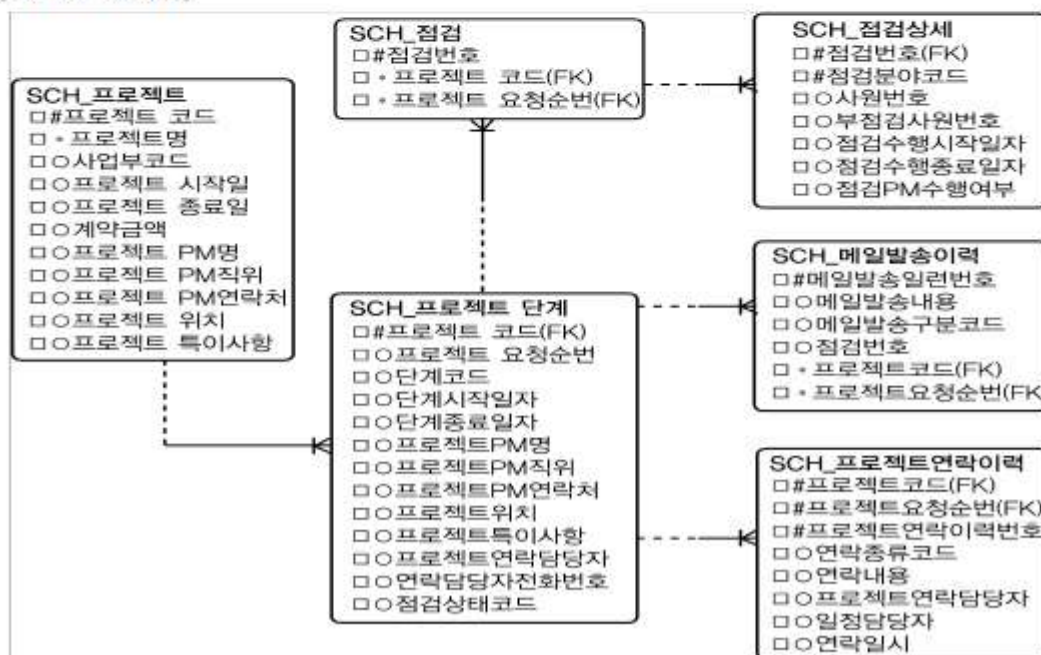
3) 식별자와 비식별자를 적용한 데이터 모델

이러한 기준에 의해 식별자와 비식별자 관계가 적절하게 설정된 데이터 모델은 [그림 1-1-54]의 사례와 같이 균형감 있게 나타난다.

[IE 표기법]



[Barker 표기법]



[그림 1-1-54] 식별자관계와 비식별자관계의 적절한 선택

상기 모델은 업무의 특성에 따라 식별자관계와 비식별자관계를 적절하게 선택함으로써 데이터 모델의 균형감을 갖추었다고 볼 수 있다.