I. 데이터 모델링의 이해

1장. 데이터 모델링의 이해

1. 데이터 모델링의 이해

1) 데이터 모델링의 정의 :
업무에 필요로 하는 데이터를 시스템 구축 방법론에 의해 분석/설계하여 정보시스템(DB)을 구축하고 개발 및 데이터 관리에 사용한다.

2) 데이터 모델링의 특징 :
- 추상화 : 현실세계를 간략하게 표현한다.
- 단순화 : 누구나 쉽게 이해하도록 표현한다.
- 명확화 : 명확하게, 한 가지 의미로 해석되도록 표현한다.

3) 좋은 모델링의 3대 요건 : 중복배제, Business Rule, 완전성

4) 관점
1. 데이터 관점 : 비즈니스 프로세스에서 사용되는 데이터를 의미
업무가 어떤 데이터와 관련이 있으며 무슨 관계인지에 대해 모델링 (What, Data)
구조 분석, 정적 분석
2. 프로세스 관점 : 비즈니스 프로세스에서 수행하는 작업을 의미
업무가 실제하고 있는 일은 무엇이며 어떻게 해야 하는지에 대해 모델링 (How, Process)
시나리오 분석, 도메인 분석, 동적 분석
3. 데이터와 프로세스의 상관 관점 : 프로세스와 데이터 간의 관계를 의미
업무를 처리하는 방법에 따라 어떤 영향을 받는지에 대해 모델링 (Interaction)
CRUD(Create Read Update Delete) 분석

5) 데이터 모델링의 단계
- 개념적 모델링 : 전사적 관점, 업무 관점 / 추상화 수준이 가장 높은 모델링 / 엔터티와 속성을 도출하고, 개념적 ERD를 작성하는 단계 / 추상화, 업무중심적, 포괄적, 개념적
- 논리적 모델링 : 특정 DB 모델에 종속된다/ 식별자를 정의하고 관계, 속성 등을 모두 표현하는 단계 / 정규화를 통해 재사용성을 높임 / 정규화, 재사용성, 신뢰성
- 물리적 모델링 : 성능, 보안, 가용성 등을 고려하여 데이터 베이스를 실제 구축 / 테이블, 인덱스, 함수 등을 생성하는 단계

6) ERD
테이블간 서로의 상관 관계를 그림으로 도식화한 것을 ERD라고 한다.
ERD의 구성 요소는, 엔터티Entity, 관계Relationship, 속성Attribute, 3가지이다.
현실 세계의 데이터는 이 3가지 구성 요소로 모두 표현 가능하다.
팀과 선수의 관계를 보자. 팀 정보와 선수 정보 사이는 물론 다른 테이블과도 이들은 어떤 의미의 연관성이나 관계를 가진다. ERD는 이와 같은 관계의 의미를 직관적으로 표현할 수 있는 좋은 수단이다.

7) 데이터 모델링의 ERD 작성 절차 (도배관술참필)
엔터티를 도출하고 그린다 → 엔터티를 배치한다 → 엔터티 간에 관계를 설정한다 → 관계를 서술한다 → 관계 참여도를 표현한다 → 관계의 필수 여부를 표현한다

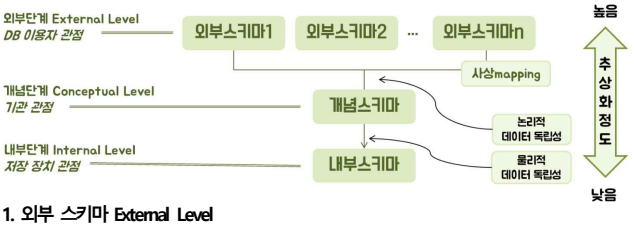
8) 데이터 모델링 과정에서의 고려사항
1. 데이터 모델의 독립성
독립성이 확보되어야 업무 변화에 능동 대응이 가능하다.
정규화를 통해 중복된 데이터를 제거해야 한다. 잘못된 데이터 모델링 이후의 변경 작업은 파급효과가 크다.
2. 고객 요구사항의 표현
데이터 정합성을 유지하며 정보요구사항을 이해하고 정확, 간결하게 표현해야 한다.
3. 데이터 품질 확보
데이터 표준을 정의하고 표준 준수율을 관리해 데이터 품질을 향상시켜야 한다.
중복/비유연성/비일관성을 주의한다.
- 중복 주의: 여러 장소에 같은 정보를 저장하지 않아야 한다.
- 비유연성 주의: 데이터의 정의를 데이터의 사용 프로세스와 분리하여 유연성을 높인다.
- 비일관성 주의: 다른 데이터의 모순된다는 고려 없이 데이터 수정을 하지 않고, 모델링 시 데이터 사이 상호연관관계에 대해 명확하게 정의하여 일관성을 높인다.

9) 데이터 독립성이란?
데이터의 독립성이란 특정 스키마를 변경해도 상위 수준의 스키마 정의에 영향을 주지 않는 성질이다. 데이터 독립성을 확보했을 때 얻는 효과는 다음과 같다.
- 데이터복잡도 증가
- 데이터 중복 제거
- 사용자 요구사항 변경에 따른 대응력 향상
- 관리 및 요지보수 비용 절감

2. 3층 스키마

1) 정의 : 사용자, 설계자, 개발자가 DB를 보는 관점에 따라 DB를 기술하고 이 관계를 정리한 ANSI 표준으로, DB의 독립성을 확보할 수 있는 방법이다.

2) 구조



1. 외부 스키마 External Level
- 사용자 개개인, DB 이용자의 관점(view 단계)으로 구성된 개인적 DB 스키마
- 응용 프로그램이 접근하는 DB를 정의

2. 개념 스키마 Conceptual Level
- 설계자 관점, 사용자 관점을 통합한 기관, 조직 전체 관점의 DB 구조
- 통합 DB 구조
- 개념 단계 하나의 개념적 스키마로 구성되어 있으며 전체 DB 내 규칙과 구조를 표현

3. 내부 스키마 Internal Level
- 개발자 관점, 저장 장치 관점 / DB의 물리적 저장 구조

3) 3층 스키마의 독립성과 매핑
각 계층 구조를 유기적으로 연결해주는 매핑을 통해, 3층 스키마는 데이터 독립성을 가진다.
특정 스키마가 바뀌더라도 매핑을 다시 하면 다른 계층 스키마 내용은 변하지 않아도 된다.
데이터독립성 보장을 위해서는 매핑하는 스크립트(DDL)를 DBA가 필요할 때마다 변경해주어야 하는데 각 단계(외부/내부/개념적 스키마)의 독립성을 보장하기 위해 변경사항이 생겼을 때 DBA가 적절한 작업을 해주기 때문에 독립성이 보장된다고도 할 수 있다.

1. 논리적 사상 = 외부적/개념적 사상 = 논리적 데이터 독립성 : 외부스키마 > 개념스키마
- 논리 스키마가 변하더라도 상위 단계에 존재하는 외부스키마가 영향을 받지 않는 성질
- 외부 적 뷰와 개념적 뷰의 상호 관련성을 정의
- 사용자가 접근하는 형식에 따라 다른 타입의 필드를 가질 수 있고 개념적 뷰의 필드 타입은 변화가 없는 것과 같다.
- 외부 인터페이스를 위하 스키마 구조는 전체 통합된 개념적 스키마와 연결된다

2. 물리적 사상 = 개념적/내부적 사상 = 물리적 데이터 독립성 : 개념 스키마 > 내부 스키마
- 데이터베이스 관리 시스템의 성능 향상을 위해 시스템을 바꾸거나 물리스키마 구조를 변경하는 경우에 물리 스키마가 바뀌더라도 개념스키마나 사용자가 이용하는 응용 프로그램은 영향을 받지 않는다
- 개념적 뷰와 저장된 DB의 상호 관련성을 정의
- 만약 저장된 DB 구조가 바뀌면 개념적 스키마는 그대로 남아있게 개념적/내부적 사상이 바뀌어야 함
- 통합된 개념적 스키마 구조와 물리적 테이블 스페이스(물리 저장된 구조)가 연결되는 구조

3. 엔터티

1) 엔터티 정의 : 엔터티는 '업무에 필요하고 유용한 정보를 저장하고 관리하기 위한 집합적인 것'이다. 엔터티는 그 집합에 속하는 개체들의 특성을 설명하는 속성을 갖는데, 이는 전체가 공유하는 공통 속성일 수도, 일부에만 해당하는 개별 속성일 수도 있다.
집합의 특성을 가지며 순수 개체이거나 행위 집합이다.

2) 엔터티 특징

1. 엔터티는 반드시 해당 업무에서 필요하고 관리하고자 하는 집합이다.

업무에서 관리하고자 하는 영역(Business Boundary)에 대한 인식이 매우 중요함

[그림 Ⅰ-1-17] 엔터티 특성-필요성

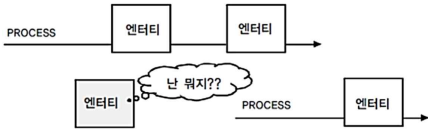
2. 유일한 식별자에 의해 식별 가능하다.

3. 두 개 이상의 인스턴스의 집합이다.

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

[그림 Ⅰ-1-19] 엔터티 특성-인스턴스 수

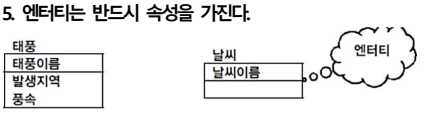
4. 엔터티는 업무 프로세스에 의해 이용된다.



업무 프로세스에 의해 이용되지 않는 엔터티는 그 업무의 엔터티가 아님

[그림 Ⅰ-1-20] 엔터티 특성-프로세스 존재

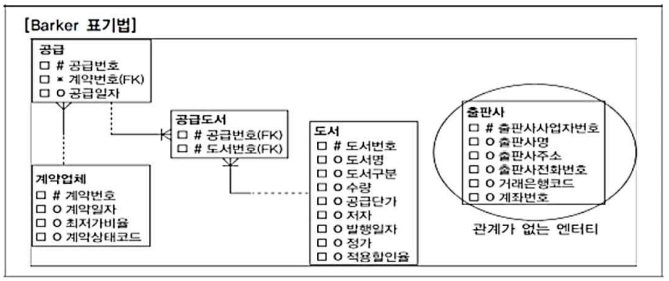
5. 엔터티는 반드시 속성을 가진다.



속성이 존재하지 않는 오브젝트는 엔터티가 될 수 없음

[그림 Ⅰ-1-21] 엔터티 특성-속성의 존재

6. 엔터티는 다른 엔터티와 꼭 관계가 있다.

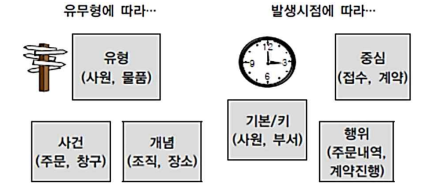


엔터티가 관계가 없으면, 잘못된 엔터티이거나 관계가 누락되었을 가능성이 큼

[그림 Ⅰ-1-22] 엔터티 특성-관계의 존재

단 관계를 생략하는 경우도 있다.
1. 통계성 엔터티 도출: 업무 진행 엔터티로부터 통계 업무만을 위해 별도로 엔터티를 다시 정의하게 되므로 관계가 생략된다.
2. 코드성 엔터티 도출: 너무 많은 엔터티와 그 관계 설정으로 인해 읽기 효율성이 저하되어 모델링 작업이 힘들 수 있다. 또 물리적 테이블과 프로그램 구현 이후에도 외부 키에 의해 참조무결성을 체크하기 위한 규칙을 DB 기능에 맡기지 않는 경우가 많아 관계를 설정할 이유가 없다.
3. 시스템 처리시 내부 필요에 의한 엔터티 도출: 트랜잭션이 업무적으로 연관된 테이블과 관계 설정이 필요하지만 업무 상 필요가 아닌 시스템 내부적 필요에 의해 엔터티이므로 관계를 생략한다.

3) 엔터티 분류



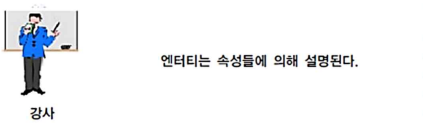
[그림 Ⅰ-1-23] 엔터티 분류

1. 유무형에 따른 분류
- 유형 엔터티Tangible Entity : 업무에서 도출되며 지속적으로 사용되는 엔터티 / 물품, 사원 등 / 물리적 형태, 지속적, 안정적, 업무도출
- 개념 엔터티Conceptual Entity : 물리적 형태는 없고 관리해야 할 개념적으로 사용되는 엔터티 / 조직, 보험상품 등 / 개념적
- 사건 엔터티Event Entity : 업무를 수행할 때, 비즈니스 프로세스를 실행할 때 발생되는 엔터티 /비교적 발생량이 많고 각종 통계 자료에 이용 / 주문, 청구, 미납 등 / 업수 발생

2. 발생시점에 따른 분류
- 기본 엔터티Basic Entity = 키 엔터티Key Entity
  다른 엔터티의 영향을 받지 않고 독립적으로 생성되는 엔터티
  타 엔터티의 부모 역할을 하며, 주식별자를 상속받지 않고 자신의 고유한 주식별자를 가진다 / 사원, 부서, 고객, 상품, 자재 등 / 독립 생성, 고유주식별자
- 중심 엔터티Main Entity
  기본 엔터티와 행위 중간에 있는 엔터티
  기본 엔터티로부터 발생되고 행위 엔터티를 생성함 / 업무처리에 중심이 된다 / 계약, 사고, 청구, 주문, 매출 등
- 행위 엔터티Active Entity
  두 개 이상의 부모 엔터티로부터 발생되는 엔터티
  업무처리를 하는 동안 발생되는 엔터티로 자주 변경되고 지속적으로 정보가 추가된다 / 보통 행위 엔터티의 데이터 양이 가장 많다. / 주문 목록, 사원변경이력 등

4. 속성

1) 속성의 정의 : 업무에서 필요로 하는 인스턴스로 의미상 더 이상 분리되지 않는 최소의 데이터 단위



[그림 Ⅰ-1-24] 속성의 정의

2) 속성의 특징
1. 속성은 해당 업무에서 필요하고 관리하고자 하는 정보이다.
2. 정규화 이론을 근거로 주식별자에 함수적으로 종속된다. 즉 기본키가 변경되면 속성 값도 변경된다.
3. 한 속성은 한 개의 값만 가진다. 한 속성에 다중값이 있을 경우 별도의 엔터티를 이용하여 분리한다.
4. 속성명은 데이터 모델에서 유일하게 사용하되, 속성값은 중복될 수 있다.

3) 속성의 분류

[그림 Ⅰ-1-27] 속성의 분류

1. 특성에 따른 분류
- 기본 속성
  비즈니스 프로세스에서 도출되는 본래의 속성
  코드성 데이터, 엔터티 식별을 위한 일련 번호, 다른 속성을 계산하거나 다른 속성의 영향을 받아 생성된 속성을 제외한 모든 속성은 기본 속성이다.
  업무로부터 분석한 속성이라도 이미 업무상 코드로 정의한 속성은 기본 속성이 아니다. / 회원 ID, 이름, 계좌번호, 주문 일자, 이자율, 원금, 예치 기간 등
- 설계 속성
  데이터 모델링 과정에서 속성을 새로 만들거나 변형하여 도출된 속성
  유일한 값을 부여한다 / 상품코드, 지점 코드, 예금분류 등
- 파생 속성
  다른 속성에 의해 만들어지는 (계산되는)속성이다 / 합계, 평균, 이자 등

2. 엔터티 구성방식(분해 여부)에 따른 분류
엔터티를 식별할 수 있는 속성을 PK 속성, 다른 엔터티와의 관계에서 포함된 속성을 FK 속성, 엔터티에 포함되고 PK/FK에 포함되지 않은 속성을 일반 속성이라 한다. 도 속성은 그 세부 의미를 쪼갤 수 있는지 에 따라 나뉜다.
- 단일 속성 : 하나의 의미로 구성된 것 / 회원 ID, 이름, 나이, 성별 등
- 복합 속성 : 여러 개의 의미가 있는 것 / 주소(시, 구, 동같은 세부 속성으로 구성) 등
- 다중값 속성 : 속성에 여러 개의 값을 가질 수 있는 것
  다중값 속성은 엔터티로 분해되므로 1차 정규화를 하거나 별도의 엔터티를 만들어 관계로 연결해야 한다 . 상품 리스트, 자동차의 색상(차 지붕, 차체, 외부의 색 별로 색이 다를 수 있음) 등

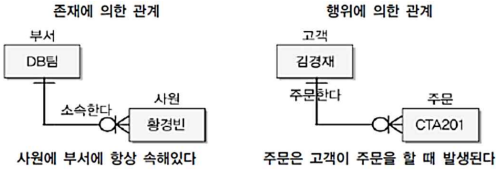
4) 엔터티, 인스턴스, 속성, 속성값의 관계
- 한 개의 엔터티는 두 개 이상의 인스턴스가, 두 개 이상의 속성을 갖는다.
  예를 들면 사원은 이름, 주소, 전화번호, 직책 등을 가질 수 있다. 사원이라는 엔터티에 속한 인스턴 스들의 성격을 구체적으로 나타내는 것이 속성이며 각각의 인스턴스는 속성의 집합이다. 한 속성은 하나의 인스턴스에만 존재할 수 있다. 속성은 관계로 기술될 수 없고 자신이 속성을 가질 수도 없다.
- 엔터티 내 하나의 인스턴스 속 한 개의 속성은 한 속성 값만 가질 수 있다.
  - 사원의 이름은 홍길동, 주소는 서울시, 전화번호가 1234, 직책이 대리라고 할 때 이름, 주소, 전화번호, 직책은 속성이고 홍길동, 서울시 1234, 대리는 속성값이다 .따라서 속성값은 각각은 엔터티가 가지는 속성들의 구체적 내용이다.
  - 속성명은 직원 엔터티의 이름, 고객 엔터티의 이름과 각 엔터티별로 동일한 속성명을 상용하여 데이터 모델의 일관성을 가지는 것이 좋다 (X)
  - 전체 데이터 모델에서 유일성을 확보하는 것이 좋다 (O)

5) 도메인이란? 도메인은 각 속성은 가질 수 있는 값의 범위다.

5. 관계

1) 관계의 정의 : 관계는 '엔터티 간의 관련성을 의미'한다. 이 관계에서 튜플의 전체 개수를 카디널리티Cardinality라고 한다.

2) 관계의 분류
관계는 연결할 때 어떤 목적으로 연결되었느냐에 따라 존재에 의한 관계/행위에 의한 관계로 구분된다.



[그림 Ⅰ-1-32] 관계의 분류

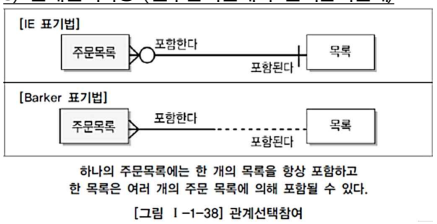
1. 존재 관계
엔터티 간의 상태를 의미한다
'소속된다'라는 의미는 행위에 의해 발생되는 의미가 아니라 황경빈 사원이 DB 팀에 소속되어 있기 때문에, 즉 존재에 의해 형성된 의미이다.
UML 클래스 다이어그램의 관계 중 연관관계이며 실선으로 표현한다. (ERD에서는 표기 구분X)

2. 행위 관계
엔터티 간에 어떤 행위가 있는 것
계좌를 개설하고 이를 통해 주문을 발주하는 것처럼 행위에 의해 관계가 형성된다. 위의 예를 보면 '주문한다'는 행위를 통해 CTA201이라는 주문번호가 생성되었으므로 이는 행위에 의한 관계이다.
UML 클래스 다이어그램의 관계 중 의존관계이며 점선으로 표현한다. (ERD에서는 표기 구분X)

3) 관계의 표기법
- 관계명: 관계의 이름
- 관계차수: 1:1, 1:M, M:N
- 관계선택사양: 필수관계, 선택관계

4) 관계차수 : 관계차수는 두 엔터티 관계에서의 참여자 수다.
- 1 : 1 관계
관계에 참여하는 각각의 엔터티는 관계를 맺는 다른 엔터티에 대해 딱 하나의 관계만 가진다.
- 1 : M 관계
관계에 참여하는 각가의 엔터티가 관계를 맺는 다른 엔터티에 대해 하나 이상의 관계를 맺는 것이다. 하 지만 반대 방향은 딱 하나의 관계만 가진다.
예를 들어 한 고객은 여러 개의 계좌를 가질 수 있지만 계좌주는 반드시 한 명이다.
- M : N 관계
두 개 엔터티가 서로 여러 개의 관계를 가지는 것이다.
관계형 DB에서 M:N 관계의 조인은 카테시안 곱이 발생한다. 그래서 두 개의 주식별자를 상속받은 관계 엔터티를 이용해 3개의 엔터티로 구분해 표현한다(M:N 관계를 1:N, N:1로 해소한다).

5) 관계선택사양 (필수참여관계와 선택참여관계)



[그림 Ⅰ-1-38] 관계선택참여

6. 엔터티 식별자

1) 정의 : 식별자는 '엔터티 내에서 인스턴스들을 구분하는 구분자로, 엔터티를 대표하는 속성이다. 따라서 하나의 엔터티에는 반드시 하나의 유일한 식별자가 존재한다.

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 발생되는 개념임
존재성	주식별자가 지정되면 반드시 데이터 값이 존재 (Null 안됨)	예) 사원번호 없는 회사직원은 있을 수 없음

2) 주식별자의 특징과 키의 종류
주식별자는 유일성과 최소성(Not Null)을 만족하는 키로, 엔터티를 대표할 수 있어야 한다. 자주 변경되지 않아야 하며 엔터티의 인스턴스를 유일하게 식별한다.
- 기본키PK : 후보키 중 엔터티를 대표할 수 있는키
- 후보키Candidate Key : 유일성과 최소성(Not Null)을 만족하는 키
- 수퍼키Super Key : 유일성은 만족하지만 최소성(Not Null)은 만족하지 않는 키
- 대체키Alternate Key : 여러 개의 후보키 중 기본키를 선정하고 남은 키

3) 식별자의 분류

- **대표성 여부에 따른 종류**
  주식별자 : 유일성과 최소성 만족. 엔터티를 대표
  보조식별자 : 유일성과 최소성 만족. 대표성은 만족x

- **생성 여부에 따른 종류**
  내부식별자 : 엔터티내부에서 스스로 생성됨. 부서코드, 주문번호
  외부식별자 : 다른 엔터티와의 관계로 만들어짐. 계좌엔터티에 회원 id

- **속성의 수에 따른 종류**
  단일식별자 : 하나의 속성으로 구성. 고객엔터티에 회원 id
  복합식별자 : 두 개 이상의 속성으로 구성

- **대체 여부에 따른 종류**
  본질식별자 : 비즈니스프로세스에서 만들어지는 식별자
  인조식별자 : 인위적으로 만들어짐. 순서번호, 주민등록번호, 사원번호, 부서코드 등

4) 주식별자 도출기준

1. 해당 업무에서 자주 이용되는 속성을 PK로 지정한다
2. 명칭, 내역 등과 같이 이름으로 기술된다면 가능하면 PK로 지정하지 않는다. 특히 '이름'은 최악(동명이인)
3. 복합으로 주식별자를 구성할 경우 너무 많은 속성이 포함되지 않도록 새로운 인조식별자를 생성한다.

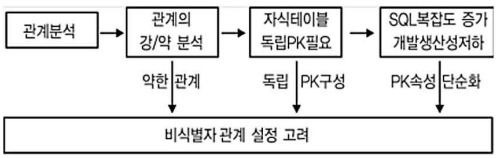
5) 식별자관계와 비식별자관계 (+강한 개체와 약한 개체)

엔터티 사이 관계 유형은 업무특징, 자식 엔터티의 주식별자 구성, SQL 전략 등에 의해 결정된다.

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	- 반드시 부모엔터티 종속 - 자식 주식별자구성에 부모 주식별자포함 필요 - 상속받은 주식별자속성을 타 엔터티에 이전 필요	- 약한 종속관계 - 자식 주식별자구성을 독립적으로 구성 - 자식 주식별자구성에 부모 주식별자 부분 필요 - 상속받은 주식별자속성을 타 엔터티에 차단 필요 - 부모쪽의 관계참여가 선택관계

1. 식별자관계 Identifying Relationship
- 부모 식별자를 자식 엔터티의 PK로 이용하는 경우에 NULL값이 오면 안되니, 반드시 부모 엔터티가 생성되어야 자기 자신의 엔터티(자식 엔터티)가 생성되는 경우다.
- 부모한테 받은 속성을 자식 엔터티가 모두 사용하고 그것만을 PK로 사용한다면 부모-자식의 관계는 1:1 관계다.
- 부모로부터 받은 속성을 포함하고 다른 부모에게 받은 속성을 포함하거나 스스로 가진 속성과 함께 PK로 지정한다면 1:M 관계가 된다.
- 이 때 식별자를 공유한 부모 엔터티를 '강한 개체 Strong Entity', 공유 받은 자식 엔터티를 '약한 개체 Weak Entity'라고 한다.

2. 비식별자관계 Non-Identifying Relationship
- 부모 엔터티로부터 속성을 받았지만 자식 엔터티의 PK로 사용하지는 않고, 일반 속성으로만 사용하는 경우이다.
- 강한 개체의 기본키를 다른 엔터티의 기본키가 아닌 일반 칼럼의 관계로 가지는 것이다.



6) 식별자 관계로만 설정할 때의 문제점

조인에 참여하는 주식별자속성의 수가 많을 경우 정확하게 조인관계를 설정하지 않고 즉, 누락하여 개발하는 경우가 생길 수도 있다.
식별자 관계만으로 연결된 데이터 모델은 주식별자 속성이 지속적으로 증가할 수 밖에 없는 구조로서 개발자 복잡성과 오류가능성을 유발시킬 수 있는 요인이 될 수 있다는 것이다.

7) 비식별자 관계로만 설정할 때의 문제점

일반적으로 각각의 엔터티에는 중요한 기준속성이 있는데 이러한 기준속성은 부모엔터티의 PK속성으로부터 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.

2장. 데이터 모델과 성능

1. 정규화

1) 정규화

정규화는 최소한의 데이터 중복, 최대한의 데이터 유연성, 데이터 일관성을 위해 데이터를 분리하는 과정이다. 따라서 데이터 중복을 제거하고 데이터 모델의 독립성을 확보한다.
정규화를 통해 업무 상 변화가 생겨도 데이터 모델의 변경을 최소화 할 수 있다.
데이터 모델이 자주 변경되는 경우 정규화하는 것이 좋다.

*** 성능을 고려한 데이터 모델링 순서 (정용트반조성)**
: 정규화 → DB용량산정 → 트랜잭션 유형파악 → 반정규화 → 이력모델 조정, PK/FK조정 등 → 성능관점에서 데이터모델을 검증

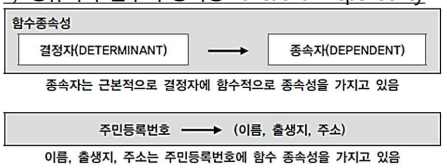
2) 정규화와 이상현상

위의 왼쪽 테이블에서 새로운 직원 엔터티가 추가된다고 가정하자. 만약 그 직원의 부서 정보가 없으면 '부서코드'를 임의로 채워넣어야 한다. 즉 불필요한 정보를 꼭 추가해야 한다.
새로운 부서 엔터티도 마찬가지인데, 사원 정보가 없다고 해도 임의의 값으로 '사원번호'를 입력하지 않으면 추가할 수 없다.
이를 이상현상이라고 하며 이럴 때 테이블 분해(정규화)가 필요하다.

3) 정규화의 절차

제 1정규화 : 속성의 원자성을 확보하며 PK를 설정 , 완전 함수 종속성 제거
제 2정규화 : PK가 2개 이상의 속성으로 이루어진 경우, 부분 함수 종속성을 제거
제 3정규화 : PK를 제외한 칼럼 간 종속성 제거 = 이행 함수 종속성 제거
BCNF : PK를 제외하고 후보키가 있는 경우, 후보키가 기본키를 종속시키면 분해한다.

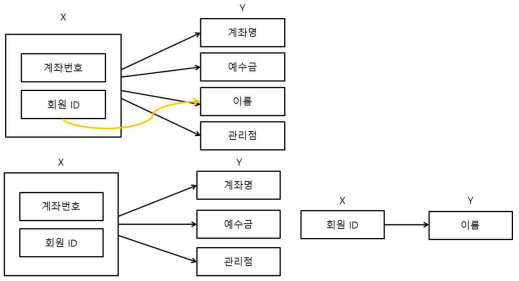
4) 정규화와 함수적 종속성 Functional Dependency



정규화는 함수적 종속성을 근거로 한다. 함수적 종속성이란 데이터들이 어떤 기준값에 의해 종속되는 현상이다. 이 때 기준값을 결정자Determinant라고 하고, 종속되는 값을 종속자 Dependent라고 한다.

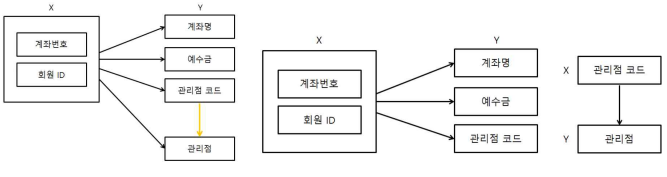
1. 제1정규화
사람 엔터티는 주민번호, 이름, 출생지, 주소라는 속성을 갖는다. 여기서 이름, 출생지, 주소라는 속성은 주민번호라는 속성에 종속된다. 어떤 사람의 주민번호가 신고되면 그 사람의 이름, 출생지, 주소가 생성되어 딱 하나의 유일한 값을 갖게 된다.
즉, 주민번호가 이름, 출생지, 주소를 함수적으로 결정한다.
기호로 표시하면, 주민번호 -> (이름, 출생지, 주소)
이 예시는 X(주민번호)가 Y(이름, 출생지, 주소)를 함수적으로 종속한다 .이 때X가 기본키가 되었다. 이렇게 기본키를 잡는 것이 제1정규화이다.
또, 이렇게 종속자가 PK에만 종속되는 것이 완전 함수 종속성이다.
릴레이션에 속한 모든 속성의 도메인이 원자 값으로만 구성되어 있으면 제1정규형에 속한다.

2. 제2정규화
제2정규화는 부분 함수 종속성을 제거하는 것이다.
부분 함수 종속성이란 PK가 2개 이상의 칼럼으로 이루어진 경우에만 발생한다. PK가 한 칼럼(위의 예시)이면 제2정규화는 생략한다.



새로운 예시를 보자. PK인 회원 ID가 변경되면 이름도 변경된다.
회원 ID가 이름을 함수적으로 종속하는 것이다. 이런 경우를 부분 함수 종속성이라고 하며 분해가 필요하다. 부분 함수 종속성을 제거하면 위와 같다. 회원이라는 새로운 테이블이 도출되고 회원 ID가 PK가 된다.

3. 제3정규화
제3정규화는 이행 함수 종속성을 제거하는 것이다.
이는 PK를 제거한 칼럼 간 종속성이 발생하는 것으로, 제1정규화와 제2정규화를 거친 뒤 제3정규화를 수행해야 한다.
주식별자를 제외한 칼럼 간 종속성에 관한 것이므로, PK와 관련성이 가장 낮다.
(좀 더 풀어서) X, Y, Z의 3개 속성에서 X->Y, Y->Z라는 종속 관계일 경우, X->Z가 성립될 때를 말한다. 아래 그림처럼 관리점이 관리점 코드에 종속되는 것이 이행 함수 종속성이다.



제3정규화를 수행하면 이렇게 관리점 테이블이 도출되고 관리점 코드가 기본키가 된다.

4. BCNF(Boyce-Codd Normalization Form)
X -> Y에서, Y가 X의 부분집합(trivial FD)이거나 X가 릴레이션 R의 수퍼키일 경우이다.

2. 정규화와 성능

1) 정규화의 문제점

정규화는 데이터 중복성을 제거한다. 그래서 데이터 모델의 유연성을 높이고 성능 향상에 도움이 된다. 하지만 데이터 조회 select시 조인을 유발한다.
그래서 CPU, 메모리 사용량이 크다. 이런 부분은 반정규화를 적용해서 해결한다.

3. 반정규화

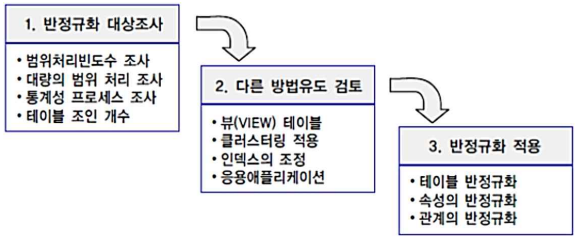
1) 반정규화

반정규화는 DB 성능 향상을 위해, 데이터 중복을 허용하고 조인을 줄이는 방법이다.
조회select 속도는 향상되지만 데이터 모델의 유연성은 낮아진다.

2) 반정규화를 수행하는 경우

- 정규화에 충실하면 종속성, 활용성은 향상되지만 수행 속도가 느려지는 경우
- 다량의 범위를 자주 처리해야 하는 경우
- 특정 범위의 데이터만 자주 처리하는 경우
- 요약/집계 정보가 자주 요구되는 경우
- 반정규화 정보에 대한 재현의 적시성으로 판단
  - 여러 테이블에 대해 다량의 조인이 필요한 경우, 적시성 확보를 위해 반정규화한다.

3) 반정규화 절차



1. 반정규화 대상 조사
범위처리빈도수 조사, 대량의 범위 처리 조사, 통계성 프로세스 조사, 테이블 조인 개수를 통해 대상을 조사한다

2. 다른 방법으로 유도 조사
반정규화 외 다른 방법(클러스터링, 뷰, 인덱스 튜닝, **파티셔닝, 응용 어플의 로직 등)이 있는지 조사한다

-클러스터링: 클러스터링 인덱스는 인덱스 정보를 저장할 때 물리적으로 정렬해 저장하는 방법으로, 인접 블록을 연속적으로 읽기 때문에 성능이 향상된다.

-파티셔닝: 논리적으로는 한 테이블이지만 여러 데이터 파일에 분산되어 저장하는 것. 데이터 조회 시 액세스 범위가 줄고, 데이터가 분할되어 있어 I/O 성능이 향상된다. 각 파티션을 독립적으로 백업/복구할 수 있다.

- 데이터값의 범위를 기준으로 하는 Range 파티셔닝
- 특정 값을 지정하는 List 파티셔닝
- 해시함수를 적용하는 Hash 파티셔닝
- 범위와 해시를 복합적으로 사용하는 Compoiste 파티셔닝

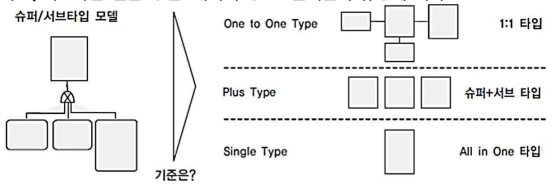
3. 반정규화 수행

4) 반정규화 기법

1. 테이블 반정규화 : 테이블 병합/분할/추가

1-1 테이블 병합
1:1 관계 테이블 병합 : 1:1 관계를 통합하여 성능 향상
1:M 관계 테이블 병합 : 1:M 관계를 통합하여 성능 향상
수퍼/서브 타입 테이블 병합 : 수퍼/서브(부모-자식) 관계를 통합하여 성능향상

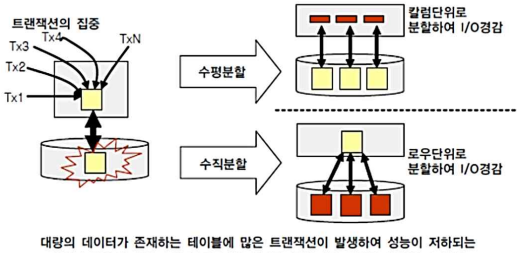
수퍼/서브타입 변환 방법: 데이터 양&트랜잭션의 유형에 따라



[그림 Ⅰ-2-25] 슈퍼타입과 서비스타입의 변환

구분	OneToOne Type	Plus Type	Single Type
특징	개별 테이블 유지	슈퍼+서브타입 테이블	하나의 테이블
확장성	우수함	보통	나쁨
조인성능	나쁨	나쁨	우수함
I/O량 성능	좋음	좋음	나쁨
관리용이성	좋지않음	좋지않음	좋음(1개)
트랜잭션 유형에 따른 성능 방법	개별 테이블로 접근이 많은 경우 선택	슈퍼+서브 형식으로 데이터를 처리하는 경우 선택	전체를 일괄적으로 처리하는 경우 선택

1-2 테이블 분할



대량의 데이터가 존재하는 테이블에 많은 트랜잭션이 발생하여 성능이 저하되는 테이블 구조에 대해 수평/수직 분할 설계를 통해 성능저하를 예방할 수 있음

[그림 Ⅰ-2-18] 테이블 수평/수직분할에 의한 성능향상

- 수직분할 : 칼럼 단위 테이블을, I/O 분산처리를 위해 테이블을 1:1로 분리하여 성능 향상
  트랜잭션이 처리되는 유형을 파악하는 것이 선행되어야 한다.
- 수평분할 : 로우 단위로 집중 발생되는 트랜잭션을 분석해 I/O 및 데이터 접근성의 효율성을 높여 성능 을 향상하기 위해 특정 값에 따라 로우 단위로 테이블을 쪼갠다.

1-3 테이블 추가
- 중복 테이블 추가 : 다른 업무거나 서버가 다른 경우 동일 테이블 구조를 중복하여 원격 조인을 제거하여 성능을 향상
- 통계 테이블 추가 : SUM, AVG 등을 미리 수행하여 계산해 둠으로써 조회 시 성능을 향상
- 이력 테이블 추가 : 이력 테이블 중 마스터 테이블에 존재하는 레코드를 중복하여 성능 향상
- 부분 테이블 추가 : 한 테이블의 전체 칼럼 중 자주 이용하는 집중화된 칼럼들의 I/O를 줄이기 위해 해당 칼럼을 모아놓은 별도의 테이블을 생성

2. 칼럼 반정규화
- 중복 칼럼 추가 : 계산에 의해 발생되는 성능 저하를 예방하기 위해 미리 값을 계산한 칼럼 추가
- 파생 칼럼 추가 : 조인을 감소시키기 위해 칼럼 중복하기
- 이력 테이블 칼럼 추가 : 불특정 조회로 인한 성능 저하 예방을 위해 이력 테이블에 기능성 칼럼(최근 값 여부, 시작과 종료일자 등) 추가
- PK에 의한 칼럼 추가 : 단일 PK 내에서 특정 값을 조회하는 경우 생기는 성능 저하를 예방하기 위해 이미 존재하는 PK 데이터를 일반 속성으로 포함하는 칼럼을 추가
- 응용 시스템 오작동을 위한 칼럼 추가 : 업무적으로는 의미가 없지만 사용자의 잘못된 데이터 처리로 원래 값 복구를 원하는 경우, 이전 데이터를 임시적으로 중복하여 보관하는 칼럼 추가

3. 관계 반정규화
- 중복 관계 추가 : 데이터 처리를 위한 여러 조인이 발생시키는 성능 저하 예방을 위해 추가적인 관계를 맺는 것

5) Row Chaining과 Row Migration

1. Row Chaining 로우체이닝
   - 로우에 데이터가 insert 후 delete 되어 한 행이 삭제되고 해당 블록에 빈 공간이 생겼을 때 새로운 데가 입력된다고 가정하자.
   - 새로운 데이터가 입력될 때 처음에 빈 공간이 있는 블록에 입력되고 그 공간이 부족할 때 새로운 블록에 나머지 데이터를 입력하는 것을 로우체이닝이라고 한다.
   - 로우 길이가 너무 길어서 데이터 블록 하나에 데이터가 모두 저장되지 않고, 두 개 이상의 블록에 결쳐 한 로우가 저장되어 있는 형태이다.
2. Row Migration 로우마이그레이션
   - 행에 입력될 수 있는 데이터 영역에 데이터가 모두 입력되어 저장 공간이 부족한 경우에서 기존 데이터의 변경 작업이 일어난다고 가정하자.
   - 변경 작업에 의해 공간이 더 필요한데 저장 공간이 없을 경우 새로운 블록으로 이동시켜 변경작업을 수행하는 것이 로우마이그레이션이다.
   - 데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에 저장하지 못하고 다른 블록의 빈 공간을 찾아 저장하는 방식이다

4. 분산 데이터베이스

1) 분산 DB

분산 DB는 DB를 연결하는 빠른 네트워크 환경을 이용해 DB를 여러 지역의 여러 노드로 위치시켜 사용성/성능 등을 극대화 시킨 DB이다.

2) 분산 DB가 효과적인 경우

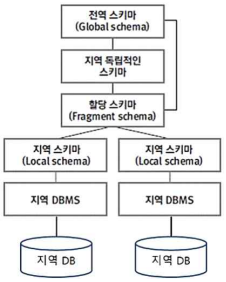
- 성능이 중요한 사이트에 적용한다.
- 공통코드, 기준정보, 마스터 데이터 등에 대해 분산환경을 구성하면 성능이 좋아진다.
  실시간 동기화가 요구되지 않을 때 좋다. 거의 실시간Near Real Time 업무일 때도 분산환경을 구성할 수 있다.
- 특정 서버에 부하가 집중이 될 때 부하 분산을 위해 쓸 수 있다.
- 백업 사이트Disaster Recovery Site를 구성할 때 간단한 분산기능을 적용하여 구성할 수 있다.

3) 분산 DB의 투명성 Transparency (위복병분장지)

- 위치 투명성 : 고객이 사용하려는 데이터 저장 장소를 명시할 필요가 없으며 고객은 데이터가 어디 있든 동일한 방법으로 데이터 접근이 가능해야 한다
- 중복 투명성 : DB 객체가 여러 시스템에 중복되어 존재해도 고객과는 무관하게 데이터 일관성이 유지 된다
- 병행 투명성 : 여러 고객의 응용 프로그램이 동시에 분산 DB에 대한 트랜잭션을 수행해도 결과에 이상이 없다
- 분할 투명성 : 고객은 하나의 논리적 릴레이션이 여러 단편으로 분할되어 각 단편의 사본이 여러 시스템에 저장되어 있음을 인식할 필요가 없다
- 장애 투명성 : DB가 분산된 각 지역의 시스템이나 통신망에 이상이 생겨도 데이터 무결성은 보장된다
- 지역 사상 투명성 : 지역 DBMS와 물적 DB 사이 사상이 보장되어 각 지역 시스템 이름과 무관한 이름을 사용할 수 있다

4) 분산 DB의 설계 방식

1. 상향식 설계 방식: 지역 스키마 -> 전역 스키마 순으로 작성
2. 하향식 설계 방식: 전역 스키마 -> 지역 스키마 순으로 작성



5) 분산 DB 장단점

장점	단점
신뢰성/가용성/효용성/융통성 high	설계/관리 복잡, 비용 high
병렬 처리 수행 -> 빠른 응답, 통신 비용 low	데이터 무결성에 대한 위협
시스템 용량 확장 easy	보안 관리/통제 어렵고 오류의 잠재성이 큼
각 지역 사용자들의 요구 수용 빠름	응답 속도 불규칙