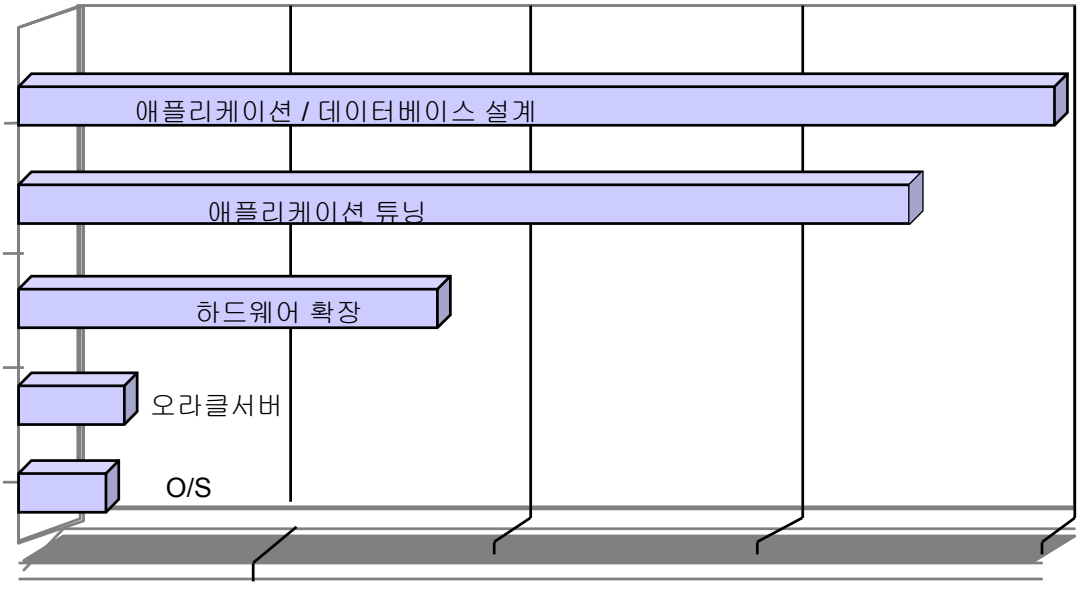
데이터 모델링/ 데이터 디자인

㈜웨어밸리 대표 컨설턴트 박태욱

개요

데이터 모델링을 할 수 있는 사람은 많다. 그러나 업무데이터의 특성과 데이터베이스의 특성을 이해하고 시스템의 이행단계에서 발생할 수 있는 문제를 분석,설계단계에서부터 미리 파악하여 사전에 이를 최대한 제거함으로써 최적화된 시스템 구현을 가능하게 하는 역할을 하는 사람은 매우 드물다. 분석, 설계시 성능의 관점에서 데이터 모델링과 업무 프로세스간의 상관관계를 파악하고 검증하는 절차는 매우 중요하다. 따라서 데이터 모델링이 어떠한 과정과 절차를 거쳐 성능에 대한 품질 보증을 할 수 있으려면 관계형 데이터베이스의 특성과 튜닝에 대한 원리를 충분히 이해하고 있는 전문가를 통하여 수행되어 져야 한다. 그렇지 않고 단순히 ERD기법에 의존하여 정규화(Normalize)나 비정규화(Denormalize)의 과정을 반복하고 오브젝트로 구성하는 절차에만 만족한다면 이는 실제 시스템 테스트 및 시스템 이행과정에서 많은 문제가 발생되는 원인이 된다. 데이터베이스 전문가라고 모두 이 분야를 잘 할 수 있는 것은 아니다. 많은 경험과 전문지식을 습득한 개인의 컨설팅 능력에 많은 부분을 의존할 수 밖에 없는 것이 현실이다. 잘 검증된 데이터 모델, 디자인은 이행단계에서 든든한 기초가 되나, 그렇지 못한 경우에는 성능향상에 많은 어려움을 겪게 하는 원인이 되므로 반드시 분석,설계단계부터 성능에 대한 관점을 유지해야만 한다. 아래의 도표는 시스템 성능에 영향을 미치는 잠재요소에 대한 중요도를 나타내고 있다.



잠재 성능개선 효과

개념

데이터 모델링은 일반적으로 논리 모델링과 물리 모델링으로 나누어져 단계별로 실행된다.

논리적 모델이란 분석단계에서 특정 업무를 하기위해 필요한 엔터티가 무엇이며 엔터티의 유일성을 보장해주는 키(key)가 무엇이며 각 엔터티간에는 어떤 상관관계가 있고 필요한 속성이 무엇인지를 설명해주는 그림이다. 이 그림의 목적은 업무담당자와 시스템 설계자 사이에서의 의사 소통이라 할 수 있다. 다시 말해 시스템과 상관없이 업무에서 필요한 데이터의 그림을 그려놓고 이를 검증하는 방법이며 주로

ERD(Entity Relationship Diagram)를 사용한다. 논리적모델에서는 액세스성능, 분산시스템, 병렬처리(Parallel Server) 등의 물리적 시스템 환경을 고려하지 않는다. 논리적 모델링 시에는 반드시 업무지식이 많은 사람의 참여가 전제가 되어야 하며, 잘 설계된 논리적 모델은 비록 업무방식이 바뀌어도 업무영역이 바뀌지 않는 한 변경이 거의 발생하지 않는다.

물리적 모델링은 논리적 모델링을 기초로 구축하고자 하는 시스템 환경(network, hardware, o/s, middleware, DB, DISK용량,분산 …)을 고려하여 수행되는 시스템의 전체적인 성능향상을 위한 기초작업으로 데이터베이스 설계의 밑그림이 된다. 논리적 모델링단계에서 물리적 모델링단계로 넘어오면서 해야 하는 작업은, 분산환경에서 테이블을 중복할 것인지 중앙에 필요한 테이블을 따로 가져갈 것인가 아니면 넷거래로 해결할 것인가? 분산환경에서 데이터의 이동시 어떻게 해결할 것인가? 병렬처리환경에서 데이터의 분할을 어떻게 할 것인가? 순환관계 엔터티를 펼칠 것인가 그대로 가져갈 것인가? 슈퍼서브타입 (Super/Sub type)관계의 엔터티를 여러 개로 분리할 것인가 하나로 할 것인가? 배타적(Arc)관계 엔터티의 관계설정을 어떻게 할 것인가? 순환(Recursive) 관계 엔터티를 몇 개의 엔터티로 할 것인가? 성능향상을 위해 테이블 중복을 허용해야 할 것인가 합병해야 할 것인가? 테이블의 컬럼을 다른 테이블에 중복할 것인가? 통계작업을 위해 합계 테이블을 몇 개로 가져가고 유일키를 무엇으로 할 것인가? 등등 시스템 환경에 따라 고려해야 할 점이 매우 다양하다.

데이터 설계는 물리적 모델을 데이터베이스의 특성에 맞게 적용하는 과정이다. 테이블 성격에 따라 어떻게 생성을 할 것인가? 인덱스는 어떻게 설정할 것인가? 스냅샷(snapshot) ,뷰(view), 프로시져(Producure) 등의 오브젝트가 필요한가? 배타적(Arc)관계 엔터티의 외부키(foreign key)를 몇 개로 할 것인가? 멀티미디어 데이터는 어떤 형태로 보관하고 액세스하게 할 것인가? 등을 구체적으로 정의하고 이를 통하여 필요한 오브젝트를 생성하게 된다. 따라서 데이터 설계시에 튜닝전문가의 관점에서 최종 검증 과정이 필요하다. 이러한 검증 과정이 없으면 때때로 애플리케이션 작성시 성능저하를 가져오는 로직이나 SQL을 구사할 수 밖에 없는 상황으로 가게 되고 실데이터 생성시 애플리케이션은 피할 수 없는 문제에 부딪치게 되어 각고의 노력을 해도 시스템 성능 향상은 미미하게 될 수도 있는 것이다.

모델링 절차

모델링은 업무 담당자와 시스템 설계자 사이의 주요한 의사소통 도구이다. 모델링은 사업의 업무영역을 명확히 하고 의사소통을 통하여 고객의 요구사항을 이끌어내고 기존의 자료와 새로운 요구자료를 충분히 검토하여 진행되어야만 하는데 명확한 설계를 하기 위해서는 일정한 방법론적인 절차를 가지고 접근할 필요가 있다.

1. 엔터티의 도출
   엔터티의 도출은 고객과의 인터뷰 내용, 기존의 문서와 파일, 기존 전산시스템의 데이터 구성등에서 찾아볼 수 있으며 엔터티인지 속성인지를 명확히 구분하여 도출되도록 하여야 할 것이다. 상위단계에서는 주요 엔터티와 엔터티간에 발생하는 주요 엔터티를 설정하는데 목표를 둔다.
2. 데이터관계(Relationship) 설정
   관계설정은 향후 전체적인 데이터 액세스 방법에 지대한 영향을 주기 때문에 업무간에 데이터 상관관계를 잘 설정하여야 한다. 관계는 엔터티간의 데이터의 일관성을 보장해 주는 장치이지만 무분별하게 사용하면 전반적으로 성능을 떨어뜨리는 요소가 될 수 있다. 상위단계에서는 M:M관계, 순환(Recursive)관계, 배타적(Arc)관계, 슈퍼서브타입(Super/Sub type) 관계 등이 많이 도출되는 특징이 있다.
3. 유일키의 설정
   주요 엔터티들의 유일키는 하위 엔터티에 영향을 주게 되며 잘못 사용시 시스템 성능에도 지대한 영향을 줄 수 있다. 따라서 유일키는 충분한 검토를 통하여 명확하게 제시함으로써 관계된 업무담당자 들이 수긍할 수 있는 후보자를 선정하여야 한다.
4. 속성의 결정

각 엔터티에 속해야 할 속성들을 모델링단계에서 최대한 찾아야 하며 이들을 어느 엔터티에 속하게 하는 것이 최적인지를 검토해야 하며, 기존에 이미 사용되던 하나의 속성을 여러 개의 속성으로 분리하거나 여러 개를 하나로 합해야 하는 경우도 발생한다. 때때로 엔터티를 속성으로 또는 속성을 엔터티로 잘못 판단한 경우에 애플리케이션 구성시 복잡한 로직을 사용하게 되는 문제를 야기시키는 원인 될 수도 있다.

5. 정규화 작업

   정규화는 5단계로 이루어져 있으나 주로 3단계까지를 적용하고 있으며 시스템환경을 고려하지 않는 논리적 모델에서는 필수적으로 이를 수행하여 데이터의 구조를 명확히 할 필요가 있다. 이는 정형화된 과정과 절차를 거쳐 시행되며 주요한 엔터티와 관계(Relationship)에서부터 시작하는 것이 효율적이다.

6. 비정규화 작업

   프로세스와 데이터 모델간의 상호검증시 엔터티의 구조를 바꾸거나 속성을 중복 또는 변경시키는 과정으로 주로 물리적인 모델링과 설계단계에서 시행되며 주 목적은 애플리케이션의 성능향상을 위한 것이며 추가로 유지,보수 관점을 포함하여 적용하는 것이 좋다.

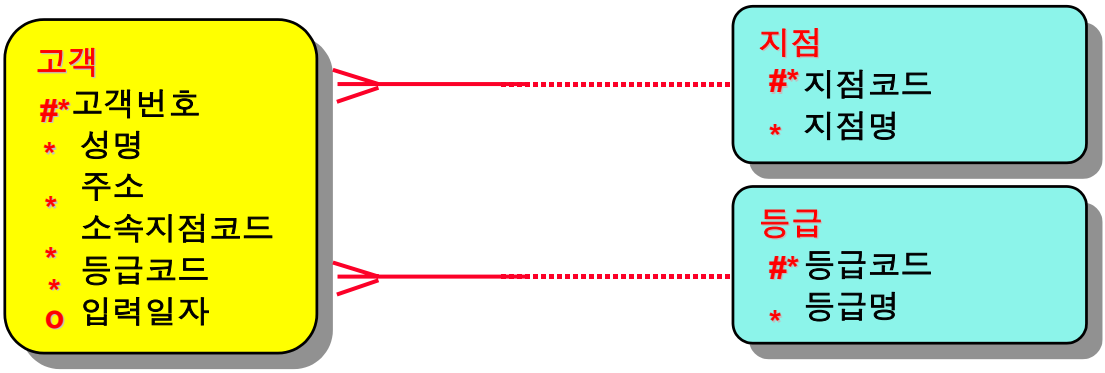
엔터티의 종류와 특성

엔터티는 데이터가 발생하거나 데이터를 액세스 하는 형태별로 몇 가지로 구분해 볼 수 있으며 형태별로 몇 가지 공통적인 특성을 가지고 있다.

■ **상위 엔터티 (Primary Entity)**

다른 엔터티 데이터의 존재여부와 관계없이 만들어지며 다른 엔터티의 데이터를 발생시키는 마스터 역할을 하는 엔터티이다. 고객, 주민정보,상품정보,사원,부서 등의 테이블이 이에 해당하며, 다음과 같은 특성을 가지고 있다.

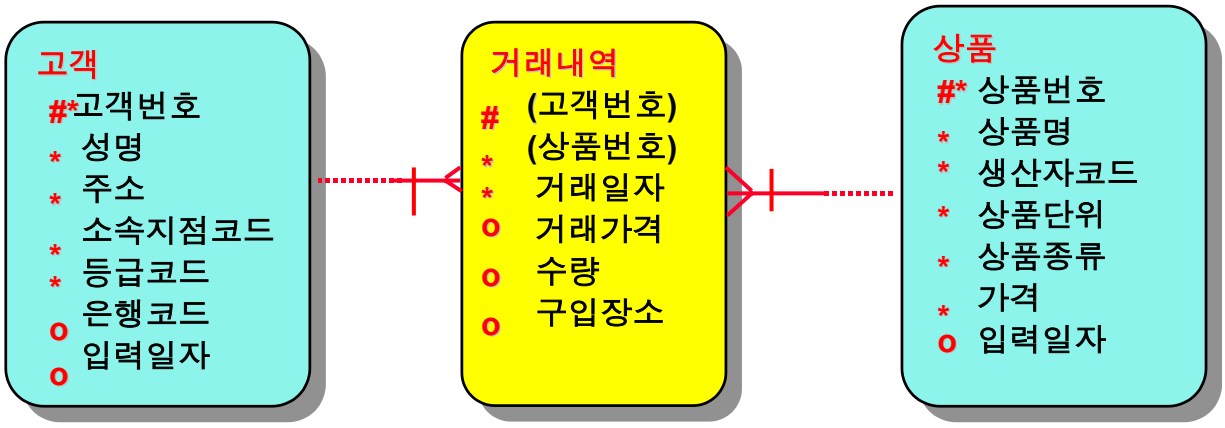
- ❖ 유일키는 보통 하나의 속성으로 만들어 지는 경우가 많다.
  예를 들면, 고객 : 고객번호, 주민 : 주민등록번호,상품 : 상품번호 등으로 되어 있다. 그러나 때로는 여러 개의 속성을 결합하여 유일키를 만드는 경우도 발생하는데 상위 엔터티의 유일키는 하위 엔터티에서 유일키 또는 외부키로 자주 사용되므로 복잡성을 제거하기 위해 일련번호(sequence)를 유일키로 사용하는 것을 고려해야 한다.
- ❖ 데이터가 급격히 늘어나지 않고 지속적으로 증가하는 특성을 가지고 있다. 따라서 테이블 생성시 저장방법은 최초값(INITIAL)은 크고 증가값(NEXT)은 적게 설정하는 것이 일반적이다.
- ❖ 외부키는 주로 코드성 엔터티의 유일키들이며 데이터의 변경이 거의 없는 코드들에 대하여는 외부키를 생략하여 사용하는 경우가 많다.
- ❖ 데이터 액세스시 주로 다른 엔터티와 조인하여 사용되며 조인시 주로 유일키를 사용하므로 액세스 유형이 단조롭고 인덱스도 간단하다. 가끔 인덱스가 유형별로 많이 필요한 경우도 발생하는데 데이터의 변경이 적으므로 시스템 성능에 영향을 끼치지는 않는다.



■ **Intersection (Associative) 엔터티**

엔터티와 엔터티의 관계가 M:M인 경우 이는 두개의 엔터티 사이에서 발생하는 거래에 의하여 발생하는 엔터티로 업무거래의 빈도에 따라 데이터가 발생한다. 계약원장, 거래내역 등이 이에 속한다.

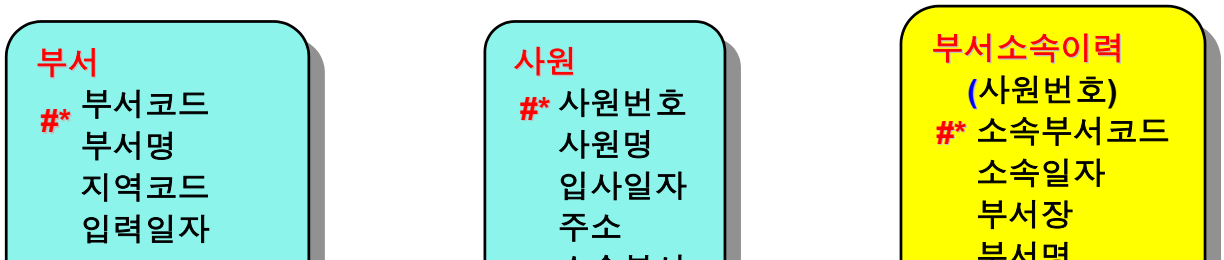
- 유일키는 2개 이상의 상위 엔터티 유일키를 결합하고 사용하거나 속성을 변형 또는 추가하여 사용한다.
  예를 들면, 보험계약 : 증권번호(삼품코드+년월+일련번호), 거래내역:상품번호+고객번호 등이다.
- 대량의 데이터가 급격히 추가,변경,삭제가 발생하는 특성을 가지고 있다. 따라서 테이블 작성시 STORAGE의 최초값(INITIAL)과 증가값(NEXT)이 모두 충분한 크기로 만들어져야 하며 데이터 재구성(Reorganization)을 주기적으로 할 필요가 있다.
- 외부키를 많이 사용하며 따라서 데이터의 변경시에 성능에 영향을 주는 요소가 된다. 따라서 참조 무결성이 불필요하다고 판단되는 외부키는 작동정지(disable) 또는 삭제하도록 한다.
- 데이터의 양도 많고 액세스 유형도 다양하며 트랜젝션의 발생빈도가 많으므로 애플리케이션의 성능에 지대한 영향을 주며 시스템의 성능과 직결된다. 따라서 데이터의 분석과 데이터 액세스 유형분석을 철저히 하여 데이터의 저장방법, 데이터 액세스방법에 따른 인덱스구성 및 SQL의 최적화를 실행하여야 한다. 필요시 데이터의 파티션을 고려하여야 한다.
- 대부분의 속성이 상위 엔터티에 없는 속성으로 이루어지며, 비정규화에 의한 컬럼도 발생이 많을 수 있다.



■ 이력 (Historical) 엔터티

하나의 엔터티에서 발생하는 이력을 관리하기 위하여 발생하는 엔터티로 과거의 어느 시점에 대한 정보제공을 목적으로 한다. 부서소속이력, 계약이력, 소유자 이력 등이 이에 해당된다.

- 유일키는 이력대상인 엔터티의 유일키에 일자와 일련번호 등을 결합하고 사용한다.
- 주기적으로 대량의 데이터를 입력하는 배치작업이 발생하는 특성을 가지고 있다. 따라서 테이블 작성시 STORAGE의 증가값(NEXT)은 일정주기에 입력되는 크기를 산정하여 사용할 수 있다. 데이터가 많은 경우 관리 목적상 데이터의 파티션을 고려할 필요가 있다.
- 외부키는 이력대상인 마스터 엔터티의 유일키를 사용하며 그 외에는 불필요한 경우가 대부분이다. 왜냐하면 외부키가 되는 속성은 대부분 마스터 엔터티에서 이미 검증이 된 데이터이기 때문이다.
- 데이터의 양은 많으나 액세스 유형이 단조롭고 트랜젝션의 발생빈도가 적으므로 시스템의 성능에 거의 영향을 주지 않으므로 인덱스 구성이 자유롭다.
- 대부분의 속성이 이력대상인 마스터 엔터티의 속성으로 이루어 진다.



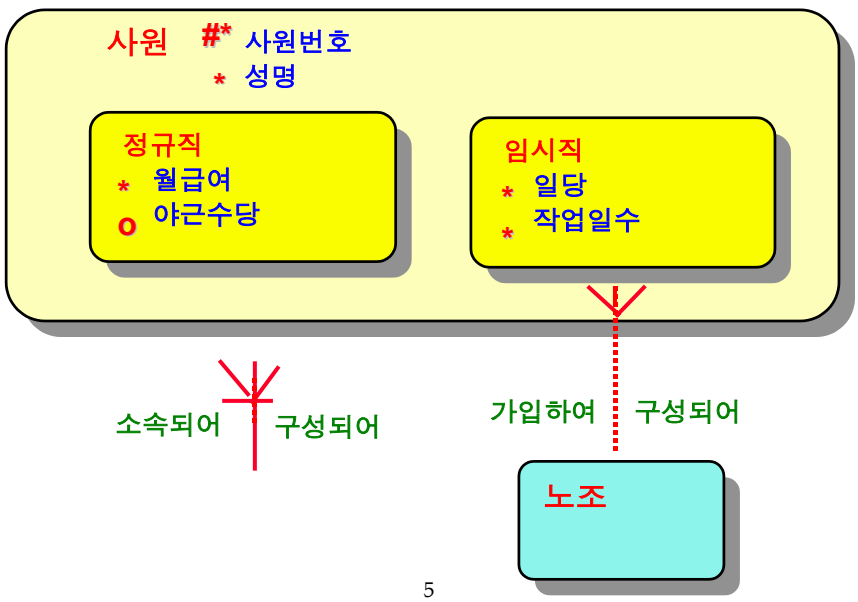
■ **Associate entity(intersetion entity)와 historical entity의 차이점과 특징**

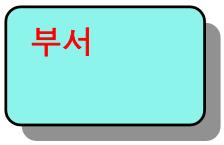
구분	Associate entity	historical entity
데이터생성 시기	거래(transaction)가 발생하는 경우	엔터티의 백업시점
상위 엔터티 갯수	2개 이상과 관계 형성	하나의 엔터티와 관계 형성
유일키	별도의 유일키 부여 또는 상위 엔터티들의 유일키 + 추가 속성	상위 엔터티 유일키+일자(일련번호)
속성	상위 엔터티와 관계없이 생성	주로 상위 엔터티의 속성 포함
데이터 발생빈도	매우 빠르게 추가, 변경됨	일괄배치에 의하여 데이터 생성
액세스 패턴	다양하다	단조롭다.
시스템 영향	매우 크며 튜닝이 필요함	특정 시간에만 영향

■ **슈퍼/서브타입(Super / Sub Type) 엔터티**

하나의 엔터티에 배타적인 서브 엔터티를 2개 이상 포함하고있는 경우이다. 각 서브 엔터티는 독립적인 속성들을 가지고 있는 동시에 공통되는 속성을 가지고 있으며 업무 프로세스에 따라서 데이터의 액세스 범위가 공통적인 부분 또는 배타적인 부분이 될 수 있다. 주로 상위 엔터티 중에서 데이터 그룹이 나누어지는 엔터티들이 그 대상이 될 수 있다.

- ❖ 유일키는 슈퍼타입 엔터티의 유일키를 사용하며 서브 엔터티는 유일키를 지정하지 않는다. 디자인 단계에서 서브타입을 각각 테이블로 독립시 주로 슈퍼타입 엔터티의 유일키를 공유하여 사용하거나 고유의 유일키를 별도로 지정할 수도 있다.
- ❖ 데이터의 생성 및 변경은 엔터티의 성격에 따라 달라지며 각 서브 엔터티를 사용하는 업무성격에 따라 데이터의 양은 유동적이다.
- ❖ 데이터의 액세스 유형에 따라 엔터티를 여러 개의 테이블로 분리하거나 하나의 테이블로 가져갈 수 있다. 대부분의 경우 테이블 파티션(Partition)을 사용하면 분리하는 효과와 하나로 합치는 두 가지의 효과를 동시에 얻을 수 있으므로 이를 적극 활용하여야 한다.
- ❖ 엔터티를 몇 개의 테이블로 할 것인가는 각 서브 엔터티의 데이터량 및 프로세스의 데이터 액세스 특징에 따라 결정되어야 한다.





슈퍼/서브 타입 엔터티의 테이블 생성 3가지 방법

- **슈퍼타입 1개와 서브타입 개수만큼 테이블로 생성하는 경우**
  - ✓ 트랜젝션이 주로 슈퍼타입 엔터티 또는 서브타입 엔터티만을 액세스 하는 비율이 높은 경우
  - ✓ 슈퍼타입의 속성이 서브타입에 비해서 적고, 슈퍼타입을 주로 전체 스캔(Full scan)하는 경우

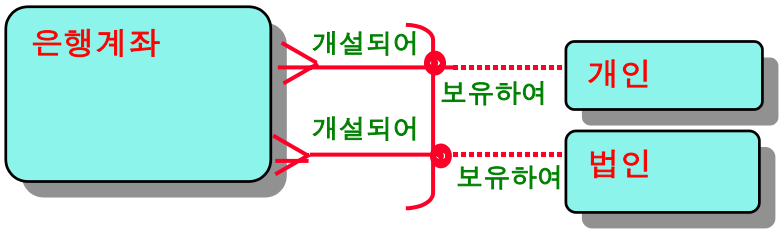
- **서브타입 개수만큼 테이블로 생성하는 경우**
  - ✓ 서로 다른 트랜젝션이 각각의 서브타입을 액세스하는 경우가 많은 경우
  - ✓ 트랜젝션이 특정한 서브타입에 치우쳐 트랜젝션이 발생하는 경우
  - ✓ 관리 목적상 (backup , analyze ….) 서브타입을 분리하는 것이 좋은 경우
  - ✓ 전체 데이터 액세스를 효율적으로 하기위해 뷰(view)를 활용하여 사용하는 것이 좋다.

- **1개의 테이블로 생성하는 경우**
  - ✓ 전체 데이터를 대상으로 배치(batch)처리하는 작업이 많은 경우
  - ✓ 하나의 트랜젝션에서 모든 서브타입을 동시에 액세스 하는 경우
  - ✓ 데이터 관리 목적상 또는 일부 데이터를 효율적으로 액세스 하기 위해서 파티션을 사용하는 것이 좋다.

■ **배타적(Arc) 관계의 엔터티**

하나의 엔터티가 2개 이상의 엔터티와 1:M으로 배타적인 관계를 갖는 경우로 아래의 예를 통하여 설명을 하고자 한다.

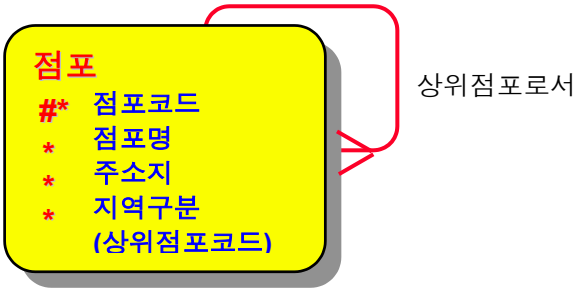
- ❖ 관계된 엔터티들의 유일키는 엔터티별로 결정되며 상호간에 관계는 없다.
- ❖ 배타적 관계를 갖는 엔터티와 슈퍼/서브 타입의 엔터티와 다른 점은 배타적관계 엔터티는 주로 1:M의 관계를 갖고 슈퍼/서브 타입의 엔터티는 1:1의 종속관계를 갖는다는 것이며, 공통점은 마스터 성격의 엔터티가 나머지 엔터티와 배타적인 관계를 갖는다는 것이다.



■ **순환관계(Recursive) 엔터티**

조직 엔터티와 같이 다단계의 계층구조를 갖는 엔터티로서 자기자신과 관계를 가지는 엔터티이다. 주로 원장에 해당하는 상위 엔터티들 중에서 발생하는 데이터가 이미 만들어진 데이터와 관계가 있는 경우이다.

- ❖ 순환관계 엔터티의 액세스는 성능을 저하시키는 주요 요인이 되는 경우가 많다. 따라서 자기 자신과 조인에 의하여 데이터를 액세스 하는 경우을 위하여 유일키와 외부키가 모두 인덱스로 만들어지는 경우가 많다.
- ❖ 계층구조가 단순한 경우 각 단계의 유일키를 나타내는 속성을 여러 개 사용하여 순환관계를 없애는 방법도 고려해야 한다. (설계시의 비정규화 고려)



■ **1:1 관계 엔터티**

본래 하나의 엔터티이나 데이터의 액세스 또는 데이터의 실제 입력여부에 따라 두개이상의 엔터티로 분리하는 경우가 발생되는데 이들은 1:1의 관계를 가지고 있다. 1:1엔터티는 매우 드물게 사용되며 데이터량 이 많지 않은 경우에는 사용하지 않는 것이 좋다.

- ❖ 데이터량이 많고 특정 속성의 크기가 매우 큰 경우 또는 특정 속성들이 거의 액세스 되지 않는 경우, 엔터티의 액세스 성능을 보장하기 위하여 위의 속성들을 별개의 엔터티로 분리하는 경우가 있다.
- ❖ 데이터의 성격에 따라 확연하게 속성들이 두개 이상의 그룹으로 나누어 지면서 그룹별로 데이터가 전부 있거나 전부 없으면서 데이터의 액세스도 별도로 일어나는 경우가 많으면 그룹별로 엔터티를 나누어 1:1관계의 엔터티로 정의할 수 있다.

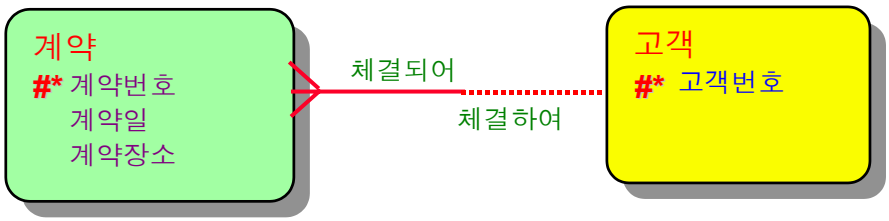
정규화 (Normalization)

정규화 작업은 데이터모델을 검증하는 방법으로 사용되며 이론적으로 5단계의 정규화 작업이 있지만 실제적으로 3단계까지를 적용하여 사용하는 경우가 대부분이며 3단계까지 적용하였다면 정규화 작업은 90%이상 실행되었다고 볼 수 있다. 정규화 작업은 논리적인 모델을 검증하는 마지막 단계로 물리적인 설계에 선행되어 작업이 이루어지며 성능을 고려하여 실행되는 비정규화와 반대되는 개념이라 할 수 있다.

■ 중복 컬럼의 삭제 (NO repeatiting group)

하나 이상의 속성들을 같은 엔터티 내에서 중복하여 사용하지 않는다. 위와 같이 하나의 고객이 여러 건의 계약을 가지는 업무인 경우 이는 속성으로 정의해서는 안되며 반드시 별도의 계약 엔터티로 표시하여야 한다.

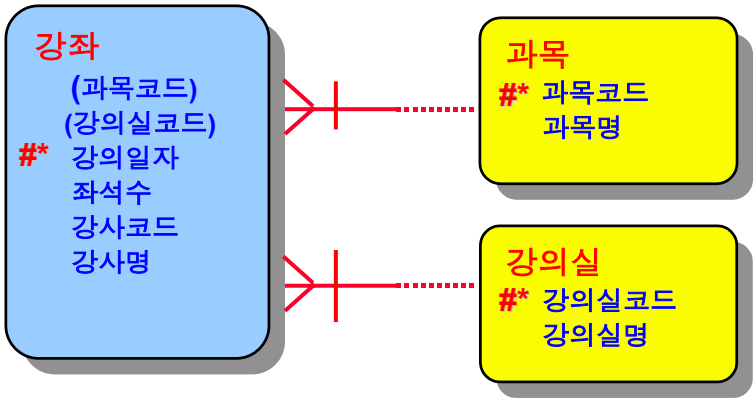
위와 같이 하나의 고객이 여러 건의 계약을 가지는 업무인 경우 이는 속성으로 정의해서는 안되며 반드시 별도의 계약 엔터티로 표시하여야 한다.



■ 유일키 전체에 대해 종속(Whole key depedent)

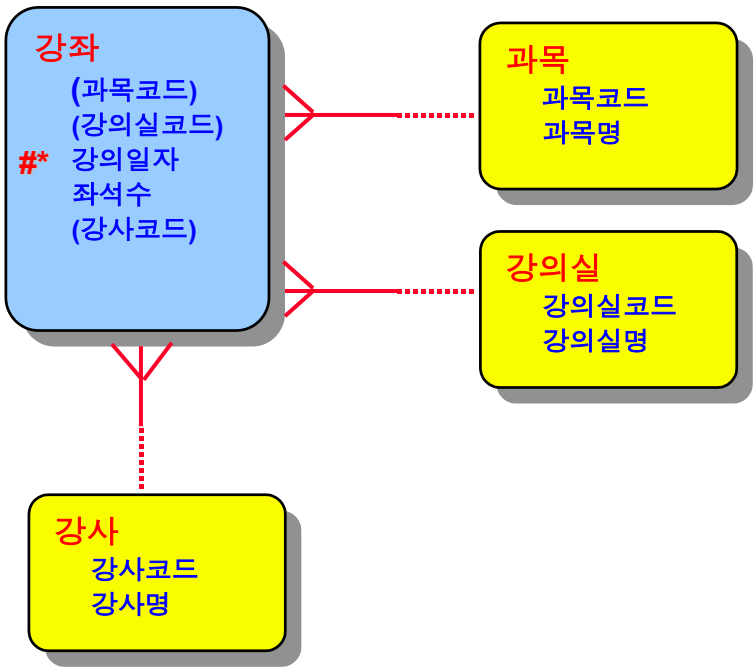
하나의 엔터티 내의 속성들은 유일키의 일부가 아닌 유일키 전체에 종속되는 것들만 존재하여야 한다. 2단계는 유일키가 여러 개의 속성들로 이루어진 경우에 실행된다.

위의 엔터티에서 과목명은 과목코드, 강의실명은 강의실코드에 종속된다. 이렇게 복합 유일키의 일부에만 종속되는 속성들은 현재의 강좌 엔터티의 속성이 아니다. 따라서 다음과 같이 변경되어야 한다.



■ 유일키가 아닌 것에 영향을 받는 속성은 분리 (Non-Key Independent)

하나의 엔터티 내의 속성들이 유일키가 아닌 속성에 의하여 영향을 받는 속성은 다른 엔터티의 속성으로 분리해야 한다. 위의 예에서 강사명은 유일키가 아닌 강사코드라는 속성에 의하여 영향을 받는다. 따라서 이는 제3정규화에 위배되며 별개의 엔터티로 구분되어야 한다.



비정규화 (Denormalization)

정규화 작업은 유지보수를 효율적이고 쉽게 해주는 반면에 설계 단계에서 하나의 엔터티가 하나의 테이블로 생성하게 되는 경우, 테이블에 대한 액세스시 과도한 조인을 발생시켜 성능을 떨어뜨리는 요인이 되기도 한다. 따라서 구현될 시스템의 성능을 고려하여 분석단계에서 실시한 정규화 작업을 역으로 비정규화 하는 작업을 실행하여야 한다.

■ **비정규화시 고려사항**

비정규화는 데이터를 액세스하는 속도를 빠르게 하기 위해서 실행된다. 그러나 복합적인 관계를 고려하지 않고 과도하게 비정규화를 할 경우에는 데이터의 변경작업으로 인하여 오히려 성능에 나쁜 결과를 가져올 수 있다. 그러므로 비정규화시 해당 테이블에 일어나는 삭제, 입력, 변경,조회 등의 트랜젝션 비율과 각각의 비용(COST)를 계산하여 얼마만큼 성능향상이 될 지를 수치화할 필요가 있다. 또한 유지,보수를 고려하여 얼마만큼의 일이 추가될 지를 감안해야 한다.

■ **비정규화의 종류**

❖ 중복 컬럼의 허용

특정테이블(계약) 액세스시 자주 조회되는 컬럼들이 다른 테이블에 있는 경우 해당 컬럼을 두개의 테이블에 중복시키는 것을 고려한다. 특히 해당 컬럼을 액세스 하기 위해서 주로 조인이 일어나고 그 외에는 거의 발생하지 않는 경우에는 중복 컬럼의 사용을 권장한다.

❖ 유도(Drived) 컬럼의 생성

합계,평균,개수 등 다른 컬럼의 값을 조합하여 결과를 얻어야 하는 트랜젝션이 많고 상대적으로 해당 테이블에 변경(DML)작업이 적은 경우, 원하는 값(합계,평균)을 가지는 컬럼을 추가하는 것을 검토한다.

❖ 테이블의 병합 및 분리
- 테이블의 컬럼 중 주로 일부만 사용되고 가끔 테이블의 전체 컬럼이 사용되는 경우는 1:1 관계를 갖는 2개의 테이블로 분리하는 것을 고려한다. 역으로 1:1 관계를 갖는 테이블이 서로 자주 조인하는 경우에는 두개의 테이블을 하나로 병합하는 것을 고려한다.
- 주종(Master – Detail)관계인 테이블의 관계수(occurrence)가 최고 5이하이고 컬럼이 매우 적은 경우이면서 두개의 테이블이 자주 조인되는 경우에는 종속(Detail)테이블을 마스터(Master) 테이블에 병합하여 중복 컬럼으로 사용하는 것을 고려한다.

❖ 분산환경에서 테이블의 중복 또는 스냅샷(Snapshot)의 사용
- 다른 노드의 테이블을 액세스(특히 Join)하는 트랜젝션이 실시간의 데이터를 요구하지 않는 경우
- OLTP에 사용되는 테이블로 OLTP트랜젝션이 실행되는 시간에 배치작업을 해야 하고, 작업이 실시간의 데이터를 요구하지 않는 경우

❖ 마스터(원장)테이블과 이력테이블의 결합
대부분 마스터와 이력은 분리하여 별개의 테이블로 관리하는 것이 유지보수를 하는데 편리하며, 배치 작업시에도 주로 유효한 현재 데이터를 가진 마스터 테이블을 대상으로 하게 되는 경우가 일반적이므로 액세스의 성능에도 도움을 준다. 그러나 때때로 이력과 마스터를 합하여 배치작업을 실행하거나 OLTP에서 이력과 마스터를 둘 다 동시에 액세스 하는 업무가 있는 경우, 두개의 테이블이 아닌 하나의 테이블에 현재와 이력데이터 모두를 관리하는 것이 효율적이 될 수 있다. 이런 경우 테이블의 키를 "기존의 유일키컬럼 + 유효시작일자 + 유효종료일자"를 사용하면 데이터 액세스시 스캔하는 범위를 최소화 할 수 있다. 이러한 경우에 단점은 애플리케이션에서 데이터를 입력/변경하는 시점에 바로 이전의 데이터를 찾아서 이를 동시에 변경해야 하는 추가적인 일이 생긴다. 따라서 데이터 액세스 성능과 데이터 관리 측면을 고려하여 결정되어야 한다.

기타 토의사항

■ **인공키(artifical primary key)의 사용**

주로 상위 엔터티에서 생성된 테이블 중에서 유일키 컬럼을 쉽게 정하기 어려운 경우가 발생하는데 그 중에서 아래와 같은 경우에 해당되는 유일키 컬럼의 경우 인공키를 사용하는 것을 검토한다.

❖ 상위 엔터티(Primary Entity)로써 키가 여러 개의 컬럼으로 이루어진 경우
상위 엔터티의 유일키는 하위엔터티와 1:M관계를 갖게 되며, 하위 엔터티 유일키의 일부가 되기도 한다. 따라서 하위 엔터티와 관계를 많이 가지는 상위 엔터티의 유일키는 간단 명료한 컬럼으로 선언되는 것이 이상적이며 그렇지 않을 경우 여러 개의 컬럼을 결합하여 유일키를 생성하는 것보다는 인공키를 사용하는 것이 애플리케이션을 작성하거나 데이터 유지보수 할 때 편리하다. 이때 인공키에 대하여 만들어지는 인덱스의 오버헤드(Overhead)는 감수해야 한다.

❖ 다른 컬럼의 값에 따라 유일키 컬럼의 의미가 다른 경우
고객 엔터티는 인공키를 많이 사용하는 대표적인 엔터티이다. 만일 고객의 종류가 개인, 법인,단체이고 개인 고객인 경우에는 주민번호+일련번호, 법인 고객일 경우에는 법인등록번호,

단체일 경우에는 일련번호를 유일키로 사용한다면 애플리케이션 작성시에 항상 이를 반영하는 로직을 추가해야 하며 데이터 유지보수에도 어려운 점이 있다. 따라서 이러한 경우에는 인공키를 사용하는 것이 여러모로 효율적으로 시스템을 구성하고 운영하는데 도움이 된다. 주로 슈퍼/서브타입(Super/Subtype) 엔터티를 하나의 테이블로 생성하는 경우가 이에 해당된다.

■ OPS(ORACLE Parallel Server) 환경을 고려한 디자인

OPS환경에서는 각각의 노드에서 설계를 잘하였어도 예기치 않는 성능저하를 가져오는 경우가 허다하다. 이는 두개 이상의 노드에서 데이터의 불일치를 방지하기 위해 PCM lock을 사용하는데, 한 노드에서 데이터의 변경이 일어나는 경우 노드간의 데이터를 일치시키려고 원하지 않는 입출력 (Ping, False Ping)이 발생하게 되고 경우에 따라 심각한 성능저하를 가져온다. 따라서 OPS환경에서는 이를 방지하기 위한 설계 검증이 반드시 필요하다.

❖ 애플리케이션 분리(partition)
같은 데이터(file)를 액세스하는 애플리케이션은 같은 노드에서 실행하도록 설계한다. 그러나 완벽하게 분리하는 것은 불가능하며 노드간에 작업분배를 고려해서 설계하여야 한다.

❖ 데이터 분리 (partition)
2개이상의 노드에서 액세스되는 테이블은 테이블 분리 또는 파티션 테이블을 이용하여 분리시켜 생성하고 각 테이블 또는 파티션을 서로 다른 파일에 저장한다. 파티션시에는 파티션 기준이되는 컬럼을 파티션키로 설정하고 파티션컬럼값의 범위로 분리하면 된다. 파티션 기준이 되는 컬럼이 없으면 추가로 구분컬럼을 생성하면 된다. (SQL사용시에는 WHERE조건문에 구분컬럼이 반드시 존재하여야 한다.) 때때로 양쪽 노드에서 같은 파티션컬럼의 값을 가지는 로우들이 액세스 되어 데이터의 완전분리가 불가능한 경우에는 해당 로우들을 자주 액세스하는 노드 쪽의 파일에 생성하게 하고 PCM lock의 개수를 늘려주도록 한다.

■ 적정한 데이터 타입의 결정

❖ **LONG, CLOB 컬럼 사용**
오라클7에서 2000byte이상의 문자는 LONG 데이터 타입을 사용하였고 그에 따라 사용상에 여러 가지 제한요소가 있었으며 성능에도 많은 영향을 주었다. 오라클8에서는 개선된 CLOB 데이터 타입을 제공하고 있으므로 이를 이용하여 데이터를 저장하고 DBMS_LOB 패키지를 사용하여 데이터를 액세스 하도록 해야 할 것이다.

LONG과 CLOB의 비교

LONG	CLOB
테이블에 한 개만 허용	테이블에 여러 개 허용
2G 까지 저장	4G 까지 저장
조회시 데이터를 return	조회시 LOCATOR를 return
In-line에 데이터 저장	In-line or Out-of-line에 데이터 저장
Object attribute가 될수 없다	Object attribute가 될 수 있다.
파티션 할 수 없다.	파티션 할 수 있다.
IOT내에 사용불가	IOT내에 사용 가능
Replication 불가	Replication 가능
Sequential 액세스	chunks단위로 액세스

PL/SQL에서 제한됨	PL/SQL에서 다양하게 사용

❖ **이진(Binary) 데이터 컬럼의 사용**

오라클7에서 2000byte이상의 문자는 LONG RAW 데이터타입을 사용하였으며, 오라클8에서는 개선된 BLOB 데이터타입을 제공하고 있고 CLOB와 마찬가지로 DBMS_LOB 패키지를 사용하여 데이터를 액세스 하여야 할 것이다.

❖ **External LOB 컬럼의 사용**

오라클8에서는 LOB데이터타입을 사용하는 Internal LOB와 BFILE을 이용하는 External LOB로 나누어진다. 대량의 이진데이터 프로세싱 위주의 시스템인 경우에는 BFILE을 사용하는 것이 좋다.

❖ **NULL 컬럼의 사용**

NULL값은 인덱스에 포함되지 않는다. 따라서 조건문에 자주 사용되는 컬럼이 NULL로 선언되어 효율적인 경우도 있으나 그 반대의 경우도 발생한다.

- SQL조건문에서 컬럼의 NULL값을 자주 체크(IS NULL)하고 조건에 해당되는 범위가 좁은 경우 인덱스 스캔이 유리함에도 불구하고 이를 사용하지 못함으로써 전체 범위 스캔(FULL SCAN)을 하게 되어 비효율이 발생한다. 이러한 경우 컬럼을 NOT NULL로 하고 Default 값을 선언하여 비효율을 방지 할 수 있다.
- NULL로 선언된 컬럼의 값이 대부분 NULL값으로 생성되고 SQL조건문에서 해당 컬럼에 대하여 NULL이 아닌 값을 체크하는 경우 인덱스를 사용하면 빠르게 액세스 할 수 있으며, 인덱스의 크기도 매우 작아서 디스크 용량도 적게 사용된다.

결론

데이터 모델링 결과인 ERD는 그 자체만으로서 품질을 평가하기가 매우 곤란하다. 그러므로 적당하게 만들어서 결과를 도출한 것과 업무 전문가와 모델링 전문가가 심혈을 기울여서 만든 것과는 외형상 비슷할 수 있다. 그러나 적당하게 만든 ERD(Entity Relationship Diagram)를 바탕으로 다음 단계가 진행되면 필연적으로 데이터 모델에 수시로 변경이 가해지고 결국에는 ERD에 관계없이 테이블이 난립하는 상황으로 갈 수도 있으며, 상위단계에서 업무 프로세스와 데이터간에 충분한 상호연관관계를 검토하지 않았다면 애플리케이션 작성시 중대한 성능문제에 봉착할 가능성이 매우 높다. 특히 관계형 데이터베이스로 시스템이 구성되는 경우에는 더더욱 그러하다. 그러므로 애플리케이션의 구현을 최우선시 하는 전시적인 시스템 구축의 구태를 벗어버리고 상위단계에서 충분한 자원을 할애해야 함을 강조한다. 업무를 분석하고 이를 데이터 중심으로 모델링을 하는 과정에 투입한 자원의 질과 양은 다음 단계인 설계와 개발, 이행 단계에서 자원을 낭비하지 않으면서 최적화된 시스템을 구현하는 밑거름이 된다는 것을 다시 한번 상기해야 할 것이다.