**데이터 모델과 성능**

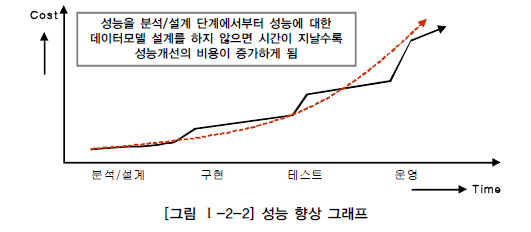
**성능 데이터 모델링의 개요**

1. **성능 데이터 모델링의 정의**
   * 데이터를 처리하는 속도는 기업의 의사결정의 속도가 빨라짐에 따라 처리하는 속도 역시 빨라져야 할 필요성을 반증해준다.
     + 성능저하 요인
       - 데이터가 대용량이 됨으로 어느정도 불가피한 성능저하 측면이 있음
       - 인덱스 특성을 충분히 고려하지 않아 무작정 인덱스를 생성해감으로 인해 성능저하
       - 데이터 모델 구조적 문제에 의한 원초적 성능저하

****

1. **성능 데이터 모델링 수행시점**

* 프록젝트 진행단계 중, 사전에 할수록 비용이 들지 않는다.
* 특히, 분석/설계 단계에서 성능데이터 모델링 수행 시 Rework Cost를 최소화 할 수 있는 **기회**를 가지게 된다.



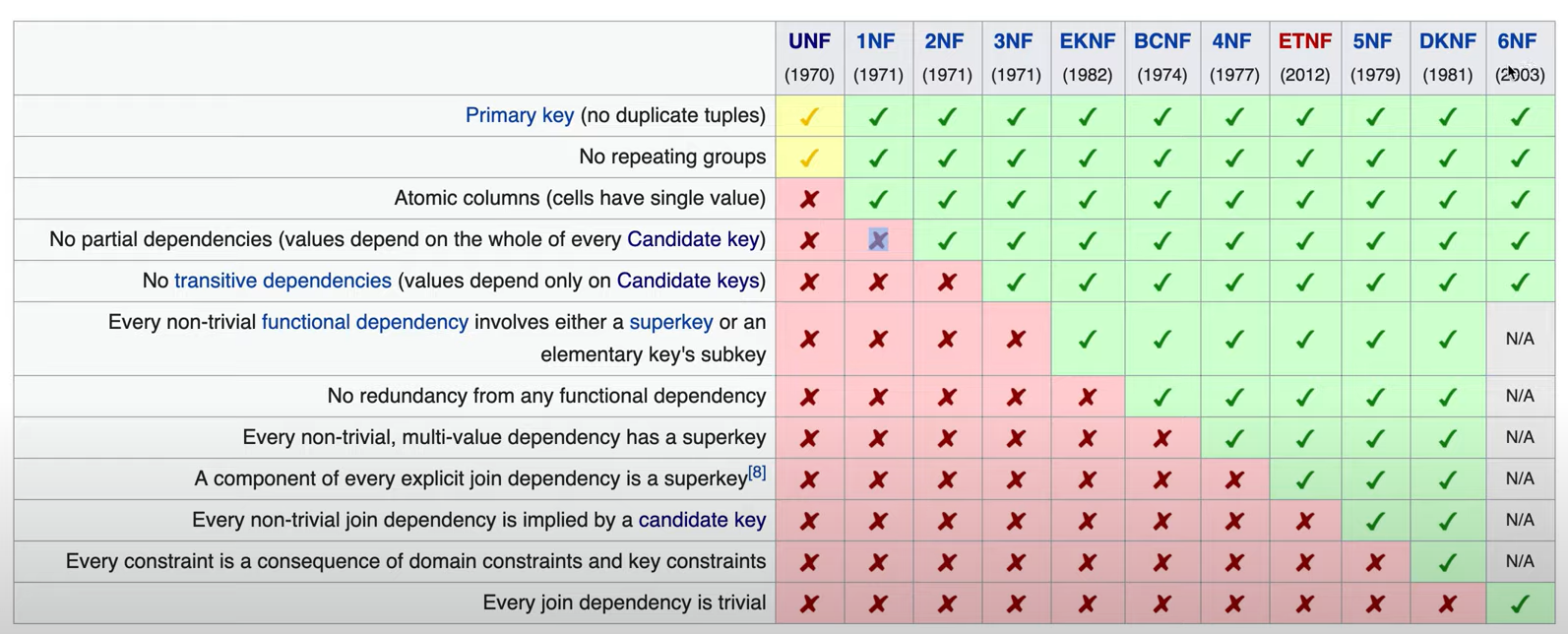
1. 성능 데이터 모델링 고려사항

* **데이터 모델링을 할 때 정규화를 정확하게 수행한다.**
* **데이터베이스 용량산정을 수행한다.**
* **데이터베이스에 발생되는 트랜잭션의 유형을 파악한다.**
* **용량과 트랜잭션의 유형에 따라 반정규화를 수행한다.**
* **이력모델의 조정, PK/FK조정, 슈퍼타입/서브타입 조정 등을 수행한다.**
* **성능관점에서 데이터 모델을 검증한다.**

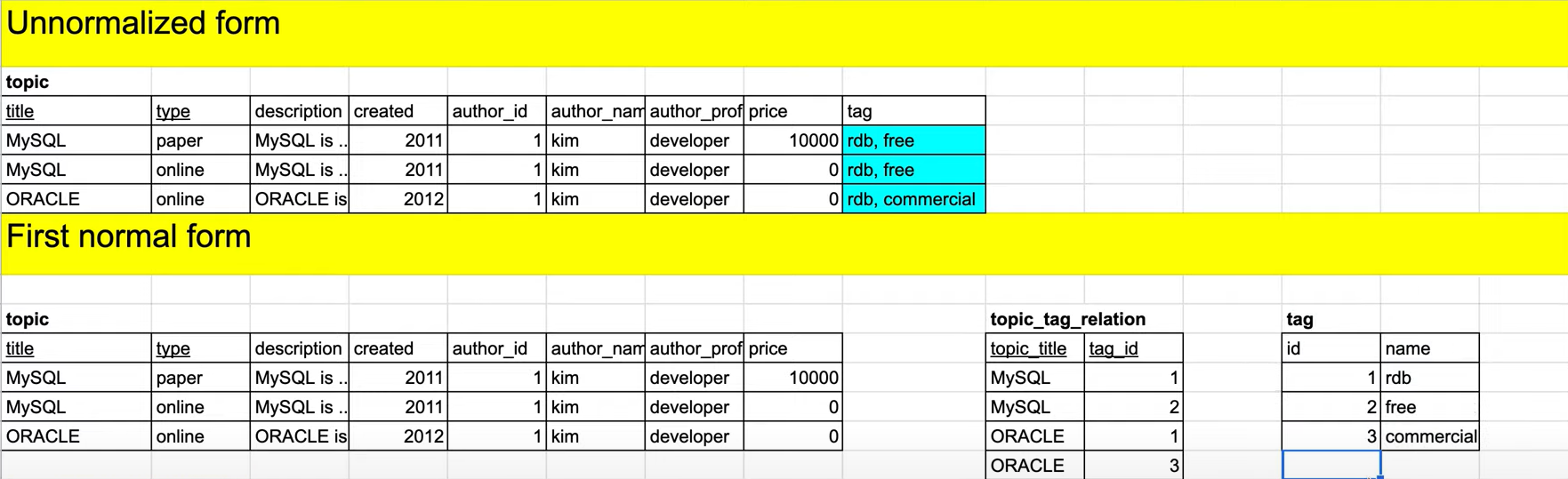
**데이터 모델과 성능**

**정규화의 성능**

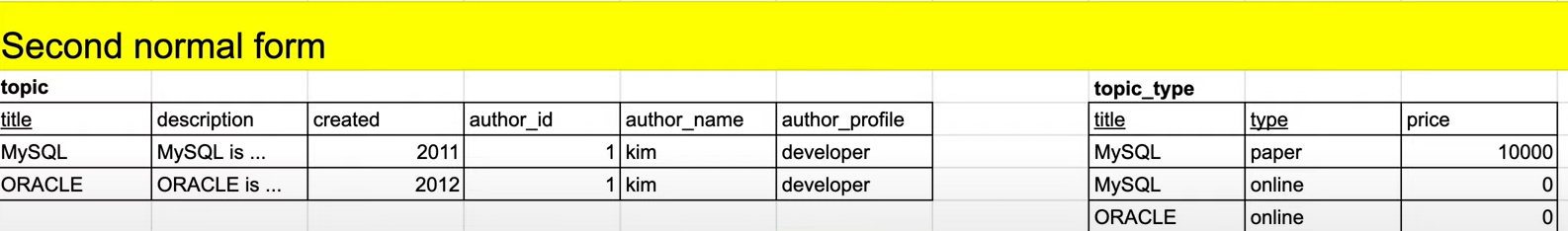
* 정규화(Normalization)



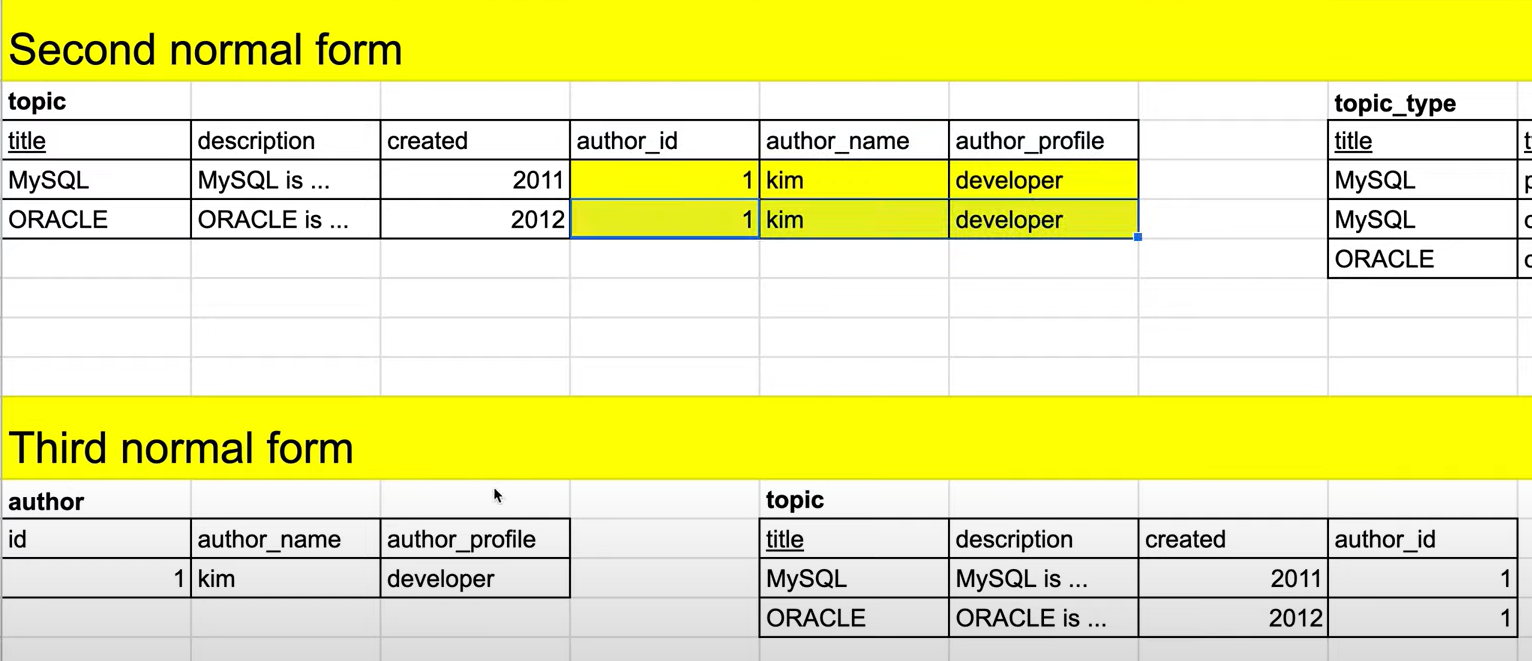
* + UnNormalizedForm 을 기준으로 1 – n Normalized Form 에 대한 조건 표
  + 실무에서는 주로 3NF까지를 주로 사용
* 제 1 정규화 : 원칙은 Atomic columns
  + 조건으로는 각 콜룸이 더 이상 쪼갤 수 없는 1개의 값만을 가져야 한다



**[제 1 정규화에 대한 과정 나열] : 핵실 내용은 Mapping Table(연결테이블)을 만드는 것이다**

****

* 제 2 정규화 : 원칙은 No Partial dependencies 즉, 부분 종속성이 없어야한다.
  + 문제점은 **부분종속성으로 인한 중복데이터가 발생한다는 것**
  + 해결책
    - 중복키를 통한 PK 중 영향을 미치는 키와 함께 따로 고유의 테이블을 생성하여 해결



* 제 3 정규화 : 원칙은 No transitive dependencies 즉, 이행적 종속성이 없음을 만족하는 정규화
  + 이행적 종속성의 의미는, 각 속성이 이행적으로 종속되어 있음을 의미한다. 예를 들어 위 경우에서는 author\_id 가 title에 의존하고 있고 autor\_name과 author\_profile은 author\_id에 의존하고 있음을 확인할 수 있다.
    - **추가적으로 만약 author\_id가 없어서 PK 및FK를 줄 수 없는 상황이고, 이행종속성을 발견했다면, 내부적으로 식별자를 생성하여 문제를 해결하면 된다.**
* ***여기부터 이론적 배경정리***

1. **정규화를 통한 성능 향상 전략**

* 기본적으로 데이터 모델링을 하면서 정규화를 하는 것은 **기본적으로 데이터에 대한 중복성을 제거**하여 주고 **데이터가 관심사별로 처리**되는 경우가 많기 때문에 성능이 향상되는 특징을 가지고 있다 \*\*\*\*\*\*\*\*
* **성능의 구분**
  + **조회성능**
  + **입력/수정/삭제 성능 두 부류로 구분된다.**

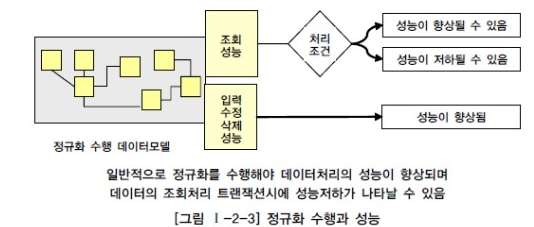
**(**두 성능이 Trade off되는 경우 많음**)**

* 정규화를 수행한다는 것은 데이터를 결정하는 **결정자**에 의해 **함수적 종속**을 가지고 있는 **일반속성**을 **의존자**로 하여 **입력/수정/삭제 이상을 제거하는 것이다**.

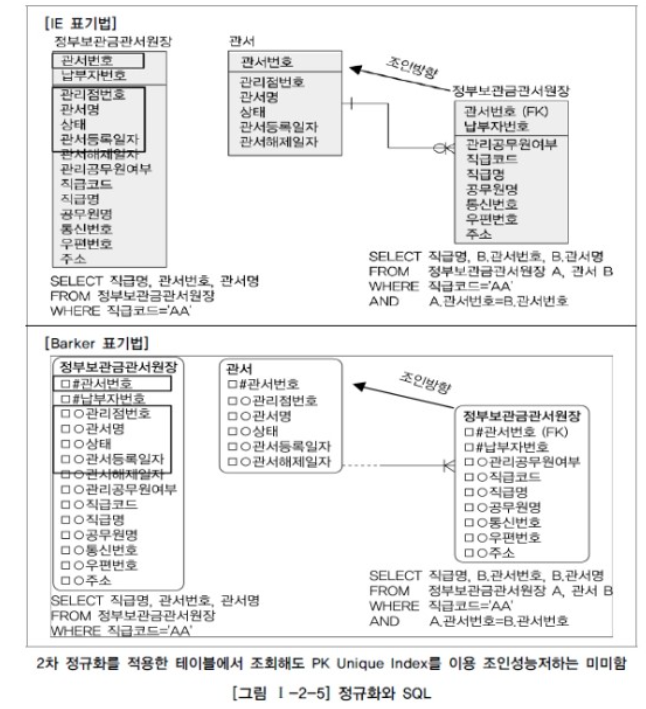
키워드 : 결정자, 함수적 종속, 일반속성을 의존자로

* 결정자에 의해 동일한 의미의 일반속성이 **하나의 테이블로 집약**되므로 **한 테이블의 데이터 용량이 최소화**되는 효과가 있다.

따라서 정규화된 테이블은 데이터를 처리할 때 속도가 빨라질 수도 있고 **느려질 수도 있는 특성이 있다,**



1. **반정규화된 테이블의 성능저하 사례 1 : 반정규화가 오히려 성능저하**

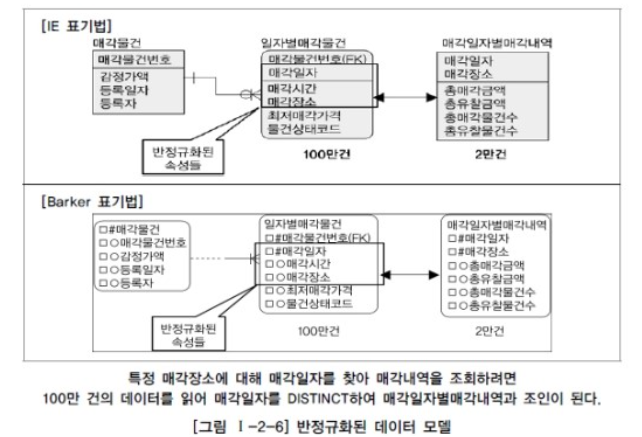
****

**왼쪽 테이블**에서는 불필요하게 **납부자번호 만큼 누적**된 데이터를 읽어서 **결과를 구분하**여 보여주어야 하지만 **오른쪽**은 **관서수 만큼만 존재하는 데이터를 읽어 곧바로 결과를 보여주기 때문이다.** 이렇게 단순한 예만 보아도 정규화를 수행하면 무조건 조회성능이 저하된다는 고정관념이 틀렸다는 것을 알 수 있다.

1. 반정규화된 테이블의 성능저하 사례 2

* Case 정의 :

**매각일자가 결정자가 되고 매각시간과 매각장소가 의존자가 되는 함수적 종속관계가 형성되는 관계이다.** 매각일자는 5천 건이 있고 일자별매각물건은 100만 건이 있는 것으로 가정하자.



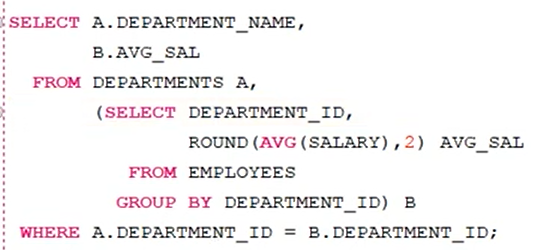
Slect b.총매각금액, b.총유찰금액

From (select distinct 매각일자 from 일자별매각물건 where 매각장소=’서울7호’), << 우선 100만건의 데이터를 읽어 Distinct 함

매각일자내역 b

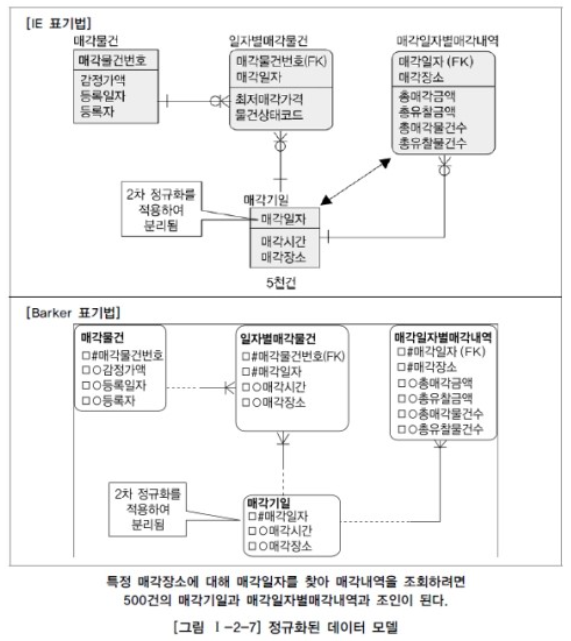
**WHERE A.매각일자 = B.매각일자**

**AND A.매각장소 = B.매각장소;**

***[+A인라인 뷰 예시]***

대량의 데이터에서 조인 조건이 되는 대상을 찾기 위해 (즉, 중복값이 있는 상태로의 결과가 아닌 고유의 결과값을 얻기 위해) 인라인뷰를 사용하기 때문에 성능이 저하된다.

위는. **복합식별자** 중에서 **일반속성이** 주식별자 속성 중 **일부에만 종속관계**를 가지고 있으므로 **2차 정규화의 대상이 된다** *(가령, 매각내역이 매각일자에만 영향을 받음)*



* 2차 정규화를 적용하여 **매각일자를 PK**로 하고 **매각시간**과 **매각장소**는 **일반속성**이 되었다.
  + 정규화 적용으로 매각내역(금액, 건수)와 관계가 연결
  + 정확한 데이터 모델링 표기 가능해짐
  + 드라이빙 된 테이블이 5천건의 테이블이 되므로 성능 향상

SELECT B.총매각금액, B.총유찰금액

FROM 매각기일 A, 매각일자별매각내역 B

WHERE A.매각장소 = ‘서울 7호’

AND A.매각일자 = B.매각일자

AND A.매각장소 = B.매각장소

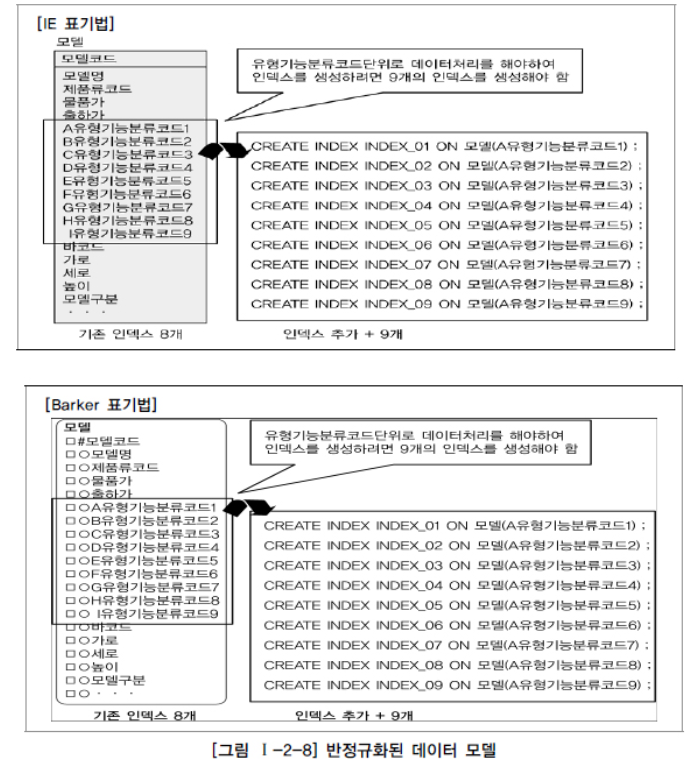
매각기일 테이블이 정규화 되면서 드라이빙이 되는 대상 테이블의 데이터가 5천 건으로 줄어들어 조회 처리가 빨라진다.

1. **반정규화된 테이블의 성능저하 사례3**

* **동일한 속성 형식을 두 개 이상**의 속성으로 나열하여 반정규화한 경우에 해당

A – I 유형의 속성 **형식이 동일하다.**

* **유형기능능분류코드 각각에 대해 인덱스를 생성해야 하므로 9개나 되는 인덱스를 추가 생성해야 한다.**

****

참고로, 한 테이블에 인덱스가 많아지면 조회 성능은 향상되지만 데이터 입력/수정/삭제 성능은 저하된다. 그래서 일반 업무처리(온라인성 업무)에서는 인덱스 수를 가급적 7~8개가 넘지 않도록 하는 것이 좋다.

**실전프로젝트에서는 어쩔 수 없이 인덱스를 생성하지 않거나 A유형기능분류코드1 하나만 인덱스를 생성하는 경우가 생긴다.** 이에 따라 A유형기능분류코드1, A유형기능분류코드2, A유형기능분류코드3...을 이용하면 SQL의 성능이 저하되는 경우가 많다.  
만약 각 유형코드별로 조건을 부여하여 모델코드와 모델명을 조회하는 SQL문장을 작성한다면 다음과 같이 나온다.

SELECT 모델코드, 모델명 FROM 모델

WHERE **( A유형기능분류코드1 = '01' )**

**OR ( B유형기능분류코드2 = '02' )**

**OR ( C유형기능분류코드3 = '07' )**

**OR ( D유형기능분류코드4 = '01' )**

**OR ( E유형기능분류코드5 = '02' )**

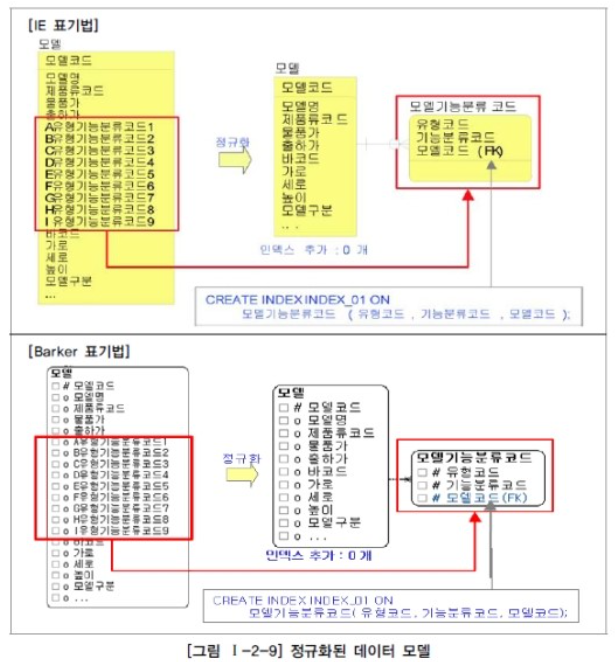
**OR ( F유형기능분류코드6 = '07' )**

**OR ( G유형기능분류코드7 = '03' )**

**OR ( H유형기능분류코드8 = '09' )**

**OR ( I 유형기능분류코드9 = '09' );**

* + **각 유형별로 모두 인덱스가 걸려 있어야 인덱스에 의해 데이터를 찾을 수 있다.**
  + **이에 대한 정규화는 아래와 같다.**

****

* + - 위는 1차 정규화이며 1차 정규화가 타당한 이유는 아래와 같다.

: 행 단위의 대상도 1차 정규화의 대상이 되지만, 열 단위의 중복 경우도 1차 정규화의 대상이 된다.

**따라서,모델에 대해 1차 정규화를 적용하면 [1-2-9]와 같은 구성을 얻을 수 있고 이에 따른 효과는 아래와 같다.**

**\*\*** 하나의 테이블에 반복적으로 나뉘어져 인덱스 생성이 어려웠던 이전과 달리 **정규화되어 분리된 이후에는 인덱스 추가 생성이 0개가 되었다.**

**\*\* 더불어, 모델기능불류코드에서 PK인덱스를 생성하여 이용함으로써 성능이 향상될 수 있다.**

**SELECT A.모델코드, A.모델명**

**FROM 모델 A, 모델기능분류코드 B**

**WHERE ( B.유형코드 = 'A'**

**AND B.기능분류코드 = '01'**

**AND A.모델코드 = B.모델코드 )**

**OR ( B.유형코드 = 'B'**

**AND B.기능분류코드 = '02'**

**AND A.모델코드 = B.모델코드 )**

**OR ( B.유형코드 = 'C'**

**AND B.기능분류코드 = '07'**

**AND A.모델코드 = B.모델코드 )**

**OR ( B.유형코드 = 'D'**

**AND B.기능분류코드 = '01'**

**AND A.모델코드 = B.모델코드 )**

**OR ( B.유형코드 = 'E'**

**AND B.기능분류코드 = '02'**

**AND A.모델코드 = B.모델코드 )**

**OR ( B.유형코드 = 'F'**

**AND B.기능분류코드 = '07'**

**AND A.모델코드 = B.모델코드 )**

**OR ( B.유형코드 = 'G'**

**AND B.기능분류코드 = '03'**

**AND A.모델코드 = B.모델코드 )**

**OR ( B.유형코드 = 'H'**

**AND B.기능분류코드 = '09'**

**AND A.모델코드 = B.모델코드 )**

**OR ( B.유형코드 = 'I'**

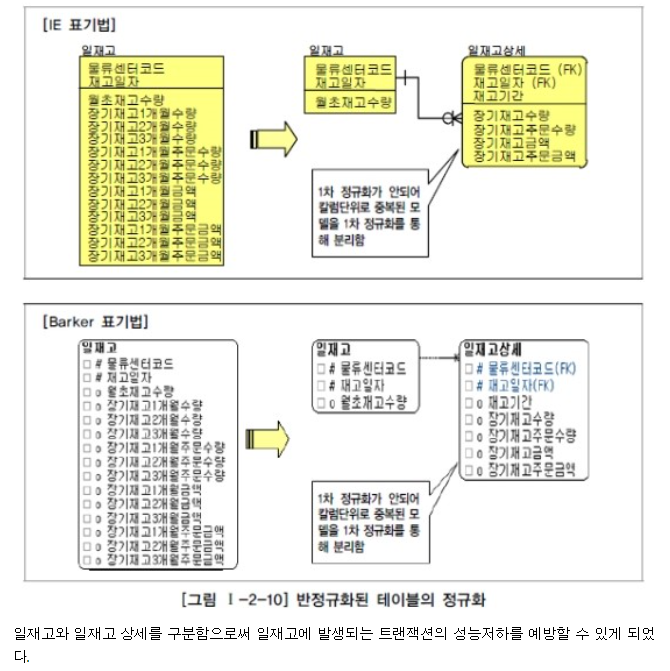
**AND B.기능분류코드 = '09'**

**AND A.모델코드 = B.모델코드 ) ;**

* **유형 + 기능 + 모델코드에 인덱스가 걸려있음**
  + **인덱스를 통해 데이터를 조회함으로써 성능이 향상 \*\*\***
    - 단수 인덱스를 통한 조회여서가 아니라 정규화를 통해 3개의 정보를 중복없이 나열 및 인덱스 공유를 통해 정확도가 올라간 것임

유의사항 : 실전 프로젝트에서 열 단위에서 중복의 경우가 자주 발생한다. 이에 대해 이에 대해, **인덱스 생성의 영향도를 파악한 이후에 적용하는 것이 좋은 방법이 된다.**

1. 반정규화된 테이블의 성능저하 사례 4



**데이터 모델과 성능**

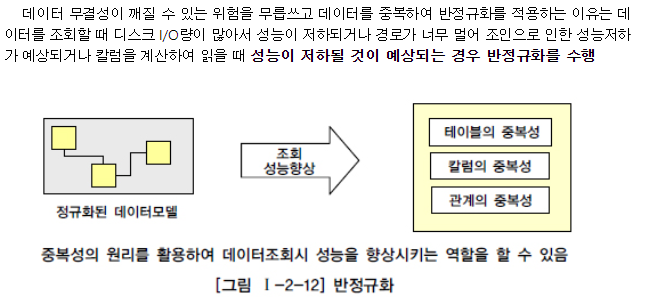
**반정규화와 성능**

1. **반정규화를 통한 성능향상 전략**

* 반정규화의 정의 : 이는, 시스템 성능향상, 개발, 그리고 운영의 다순화가 목적이며,

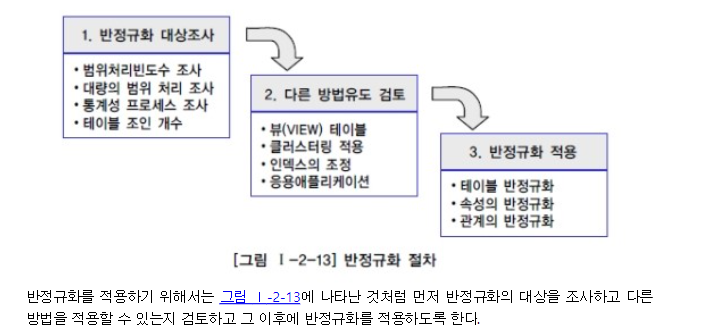
**단순화를 위해**, 오히려 중복, 통합 분리 등을 수행하는 데이터 모델링의 기법을 의미

* **협의의 반정규화 : 오히려 중복하여 성능을 향상시키기 위한 기법**
  + - **오히려, 정규화된 데이터 모델에서 중복, 통합, 분리 등을 수행하는 모든 과정을 의미**
* **테이블, 칼럼, 관계의 중복성**



* + 기본적으로 정규화는 입력/수정/삭제 뿐 아닌 조화에 대한 성능 향상의 역할을 감당한다.
  + **그러나,** 정규화만 고집한다면, **엔터티의 개수**가 증가하는 문제, **관계**가 많아져서 **비식별자 관계일 경우** **조인과정**에서 **필터링 해야 할 데이터를 가져**오는 경우가 생길 수 있다.
    - 통상, 프로젝트 설계단계에서 반정규화를 고려하게 된다. 이 때 기술적 수행이 중요하다.

**\*\*\* 반정규화 적용방법 \*\*\***

****

1. **반정규화 대상조사**
   * + - 자주 사용되는 테이블, 이에 Acess하는 Process의 수가 많으면서도, 단지 일정한 범위만을 조회하는 경우라면 반정규화를 검토해볼 것
       - 대량의 데이터 처리에 대해, **처리의 범위를 일정하게 줄이지 않으면 성능을 보장할 수 없을 경우**에 반정규화를 검토한다.
       - 별도의 통계테이블(반정규화 테이블)을 생성하는 경우
       - 테이블에 지나치게 많은 조인이 걸려 데이터를 조회하는 작업이 기술적으로 어려울 경우 반정규화를 검토
2. **반정규화 대상에 대해 다른 방법으로 처리할 수 있는지 검토**

**전제 :** 가급적이면 데이터를 중복하여 **데이터 무결성**을 깨뜨릴 수 있는 위험을 제어하기 위함

* + 지나치게 많은 조인에 대하여는, 뷰(View)를 사용하면 이를 해결할 수도 있다.

View 그 자체가 조회의 성능을 향상시키는 역할을 수행하는 것은 아니나,

개발자가 성능을 어떻게 고민하느냐에 따라 유용할 수 있음

* + 클러스터링을 적용하거나, 인덱스를 조정함으로써 성능을 향상시킬 수 있다.

**전제1 : 클러스터의 개념 : 인덱스 키 값을 기준으로 데이터를 물리적 정렬**

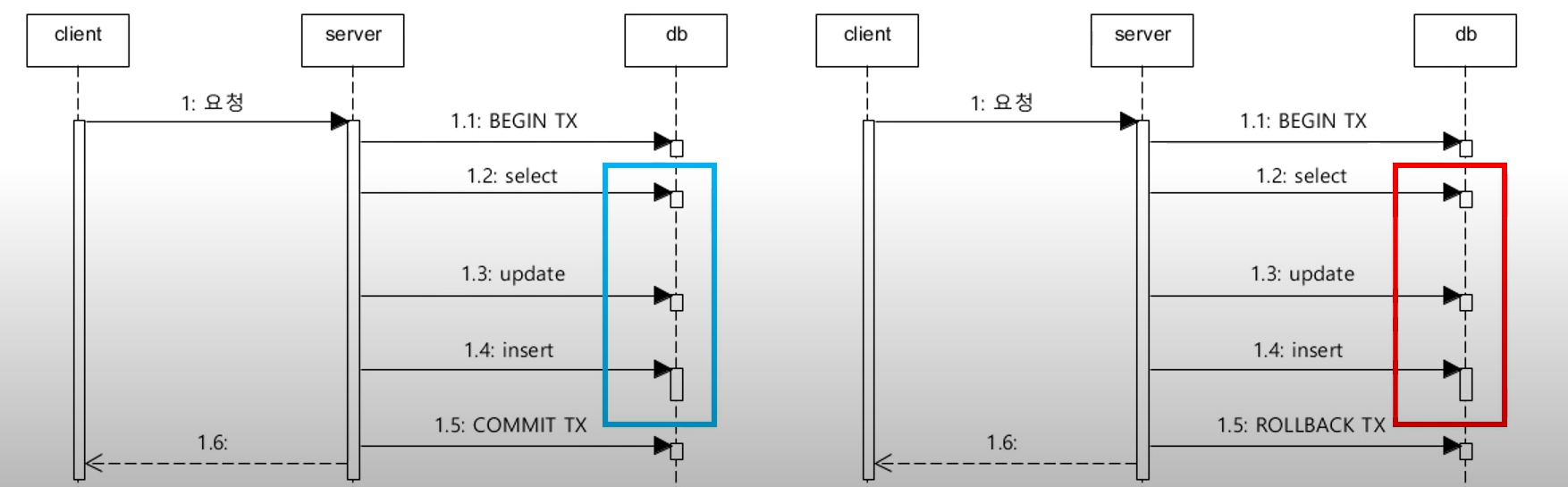
단 클러스터링을 통한 성능개선은 입력/수정/삭제하는 경우의 성능은 많이 저하된다. **이런 이유로 조화중심의 테이블이 아니라면 생성하면 안되는 오브젝트이다.**

**전제2 : 조화가 대부분이고 인덱스를 통한 성능향상이 불가능하다면, 클러스터링을 고려**

* + **PK의 성격에 따라 부분적인 테이블로 분리. 즉. 파티셔닝 기법(Partitioning)이 적용되어 성능저하를 방지할 수 있다.**
    - **인위적인 테이블을 통합 / 분리 X**
    - **물리적인 저장기법에 따라 성능을 향상 O**

**전제 : 특정 기준에 의해 물리적인 저장공간이 구분될 수 있고 트랜잭션이 들어올 때 일정한 기준에 의해 들어온다면 ~**

트렌젝션에 대한 간단요약

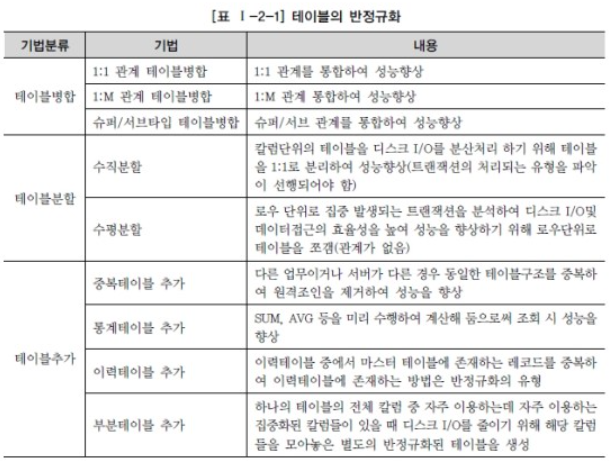


* **여러 읽기 쓰기를 논리적으로 하나로 묶음**
* **트렌잭션 시작은 , 쿼리실행 및 커밋, 롤백 시작 (반영 또는 미반영 모두 포함)**
  + **응용 애플리케이션**에서 **로직을 구사**하는 방법을 변경함으로써 성능을 향상시킬 수 있다.

**응용 메모리 영역**에 데이터를 처리하기 위한 값을 **캐쉬**한다든지 중간 **클래스 영역**에 데이터를 **캐쉬**하여 공유하게 하여 성능을 향상 시키는 것도 성능을 향상시키는 방법이 될 수 있다.

1. **반정규화 적용단계 : 3가지 추가로 고려**
   * 반정규화는 테이블 속성 관계에 대해 적용할 수 있다.
   * 중복만이 반정규화는 아니며, 테이블, 속성, 관계를 추가/분할/제거할 수도 있다.
2. **반정규화의 기법**

* **테이블 반정규화**

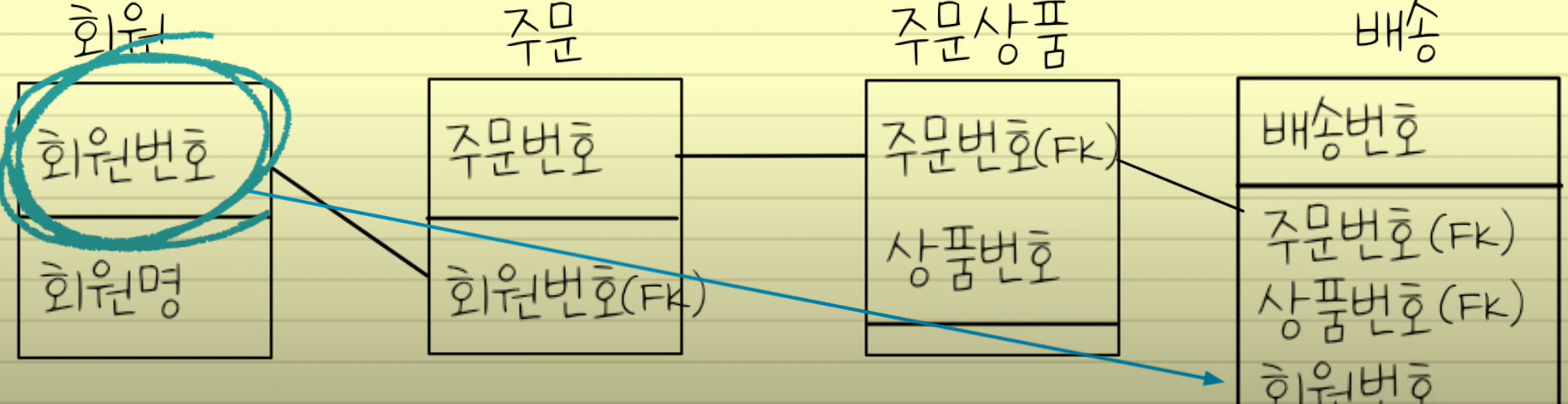
****

**세부내용**

* + **테이블병합 :** 비즈니스 로직 상 Join 되는 경우가 많아 통합하는 것이 성능 측면에서 유리하라 경우 고려
  + **테이블추가**
    - **중복테이블추가 :** 타 업무 또는 타 서버에 있는 테이블과 동일한 구조의 테이블 추가, 원격 Join 방지를 위함
    - **통계테이블추가 :** 주문번호, 상품코드 등에 따른 일일주문수량, 일일주문금액 등의 평균 통계값을 미리 계산해서 저장하는 테이블 추가
    - **이력테이블추가 :** 마스터 테이블에 존재하는 row를 트랜잭션 발생 시점에 따라 복사해두는 테이블 추가 (commit, rollback…)
    - **부분테이블추가 :** 자주 조회하는 컬럼들만 별도로 모아놓은 테이블 추가
* **컬럼의 반정규화**

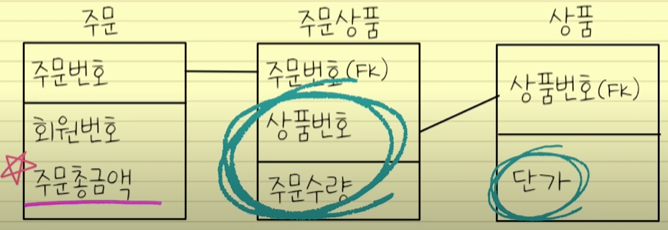
****

* + **중복칼럼 추가 :** 
    - Join Process를 줄이기 위해 중복 칼럼 추가
    - SELECT 비용은 감소하나 UPDATE 비용은 중가

****

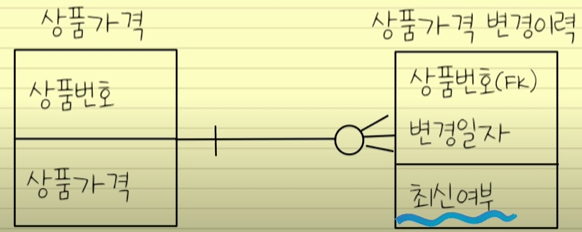
☞ 회원번호를 필두로 JOIN을 반복화 하도록 진행하기 보다는 Table 설계 시 중복칼럼을 허용하여 관리

* + **파생칼럼 추가 :**
    - Derived Column 이라고 함
    - 계산을 통해 얻어지는 결과값을 테이블에 컬럼으로 저장

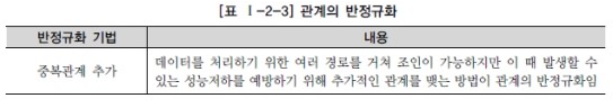


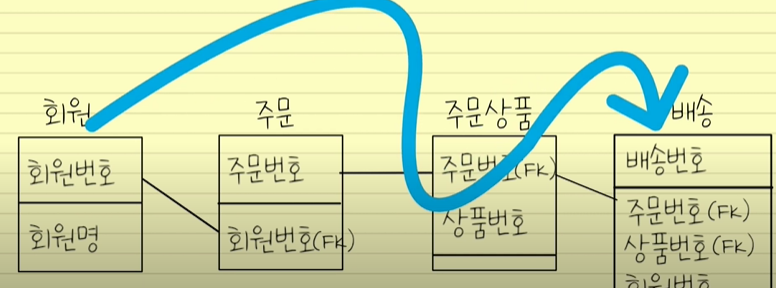
☞ 주문총액의 경우 상품에 따른 단가에 의해 판매가가 결정되면 판매 수량에 의한 주문 총 금액을 **계산**해야 하지만 그 데이터가 반복적이고 많은 수의 데이터에 의해 결정된다면, 실적에 대한 정보 같은 경우 미리 파생칼럼을 추가하여 관리하는 편이 효율적일 수 있다.

* + **이력테이블 컬럼 추가 :**
    - 이력테이블에 기능성 칼럼 추가(최신여부, 시작일/종료일 등)



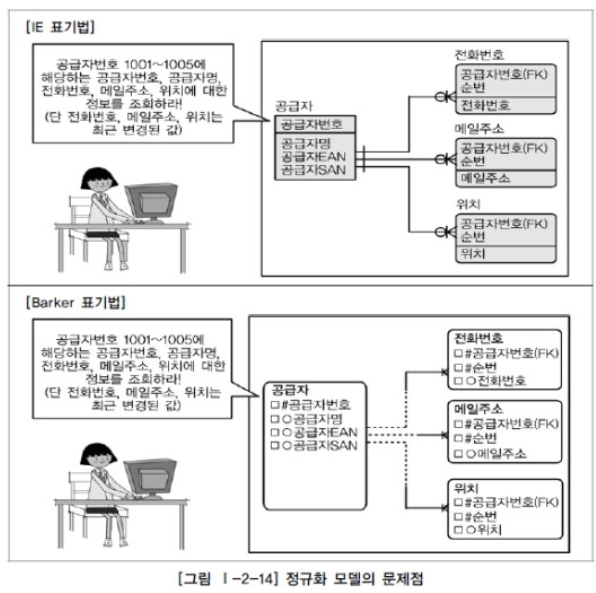
* + **PK에 의한 컬럼 추가 :**
    - **복합의미를 갖는 PK를 단일속성으로 구성하였을 경우,** 단일 PK안에서의 특정값 조회의 경우 성능 저하가 일어날 수 있음이 원이
    - **성능향상을 위해 일반속성으로 포함하는 것**
* 관계 반정규화
  + **중복관계추가 :**





* + 테이블, 칼럼의 반정규화는 데이터 무결성에 영향을 미침
  + 반면, 관계의 반정규화는 데이터 무결성을 깨뜨릴 위험을 갖지 않고서도 데이터 처리 성능을 향상시킴

1. **정규화가 잘 정의된 데이터 모델에서 성능이 저하될 수 있는 경우**

****

* + 공급자와 전화번호, 메일주소, 위치의 경우 1:M 관계이며,
  + **가장 최근의 값을 가져오기 위해서는 조금 복잡한 조인이 발생될 수 밖에 없다.**

**SQL 문 작성 예시)**

SELECT A.공급자명, B.전화번호, C.메일주소, D.위치

  FROM 공급자 A,

       (**SELECT X.공급자번호,  X.전화번호**

**FROM 전화번호 X,**

**(SELECT 공급자번호, MAX(순번) 순번**

**FROM 전화번호**

**WHERE 공급자번호 BETWEEN '1001' AND '1005'**

**GROUP BY 공급자번호) Y**

**WHERE X.공급자번호 = Y.공급자번호**

**AND X.순번 = Y.순번) B,**

전화번호 테이블과 **X**

전화번호 테이블 중 공급자 번호가 1001 – 1005 사이인 데이터 중 순번이 가장 최근인 데이터에 대하여 **Y**

Key인 공급자번호와, 조건인 순번이 동일한 데이터에 한하여 공급자번호 및 전화번호 추출 **B(B 최종)**

       (SELECT X.공급자번호,  X.메일주소

          FROM 메일주소 X,

               (SELECT 공급자번호, MAX(순번) 순번

                  FROM 메일주소

                 WHERE 공급자번호 BETWEEN '1001' AND '1005'

                 GROUP BY 공급자번호) Y

         WHERE X.공급자번호 = Y.공급자번호

           AND X.순번 = Y.순번) C,

       (SELECT X.공급자번호,  X.위치

          FROM 위치 X,

               (SELECT 공급자번호,  MAX(순번) 순번

                  FROM 위치

                 WHERE 공급자번호 BETWEEN '1001' AND '1005'

                 GROUP BY 공급자번호) Y

         WHERE X.공급자번호 = Y.공급자번호

           AND X.순번 = Y.순번) D

 WHERE A.공급자번호 = B.공급자번호

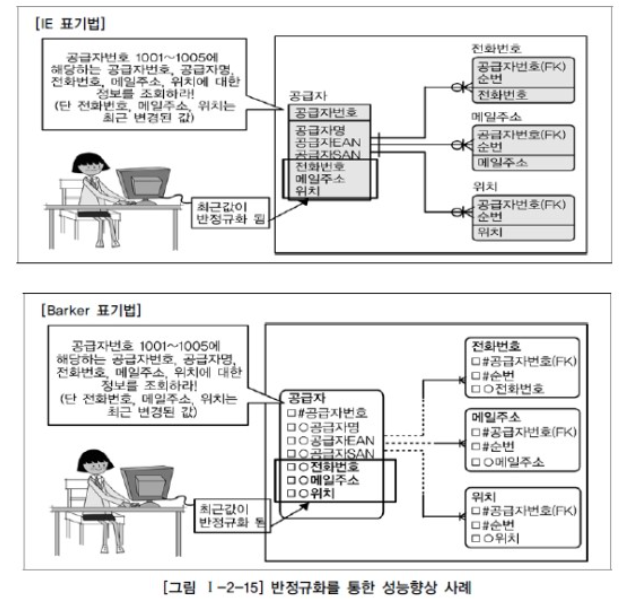
   AND A.공급자번호 = C.공급자번호

   AND A.공급자번호 = D.공급자번호

   AND A.공급자번호 BETWEEN '1001' AND '1005';

**정규화 된 모델이 적절하게 반정규화 되지 않으면 위와 같은 복잡한 SQL구문은 쉽게 파생될 수 있다.**

**해결책으로, 가장 최근에 변경된 값을 마스터에 위치시키면, 쿼리 구문이 간단해질 수 있다.**

****

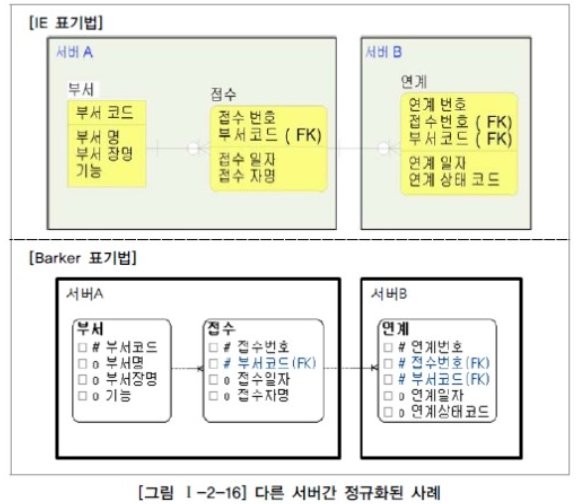
**SELECT 공급자명, 전화번호, 메일주소, 위치**

**FROM 공급자**

**WHERE 공급자번호 BETWEEN '1001' AND '1005'**

1. **정규화가 잘 정의된 데이터 모델에서 성능이 저하될 수 있는 경우**

**데이터베이스서버가 분리 되어 분산데이터베이스가 구성되어 있을 때 빤정규화를 통해 성능을 향상시킬 수 있는 경우**

****

* + **서버B**에서 데이터를 조회할 때 **빈번하게 조회되는 부서번호가 서버A에 존재**하기 때문에 연계, 접수, 부서 테이블이 모두 조인이 걸리게 된다.
  + 게다가 분산데이터베이스 환경이기 때문에 다른 서버간에도 조인이 걸리게 되어 성능이 저하되는 것이다.

**반정규화 전 SQL 문 예시**

Select c.부서명, a. 연계코드

From 연계 a, 부서 c

**Where a.부서코드 = b.부서코드**

**And a.접수번호 = b. 접수번호**

**And b. 부서코드 = a. 부서코드 <<**

a 예계 테이블부터 c 부서 테이블까지 순차적으로 서버 조인을 통계 데이터를 연결시키는 과정이다

And a.연계일자 between '20040801' AND '20040901';

\*\*\* 서버A에 있는 부서테이블의 부서명을 서버B의 연계테이블에 부서명으로 속성 반정규화를 함으로써 조회 성능을 향상시킬 수 있다. \*\*\*

**반정규화 후 SQL 문 예시**

SELECT 부서명, 연계상태코드

  FROM 연계

 WHERE 연계일자 BETWEEN '20040801' AND '20040901';

SQL구문도 간단해지고 분산되어 있는 서버간에도 DB LINK 조인이 발생하지 않아 성능이 개선되었다.

반정규화를 적용할 때 기억해야 할 내용은 **데이터를 입력, 수정, 삭제할 때는 성능이 떨어지는 점**을 기억해야 하고 **데이터의 무결성 유지에 주의를 해야 한다.**

**데이터 모델과 성능**

**대량데이터에 따른 성능**

1. **대량 데이터발생에 따른 테이블 분할 개요**
   * 테이블 분할을 통해 대량 데이터 발생 시 성능 저하를 예방한다 \*\*\*\*
     + I/O 경감을 위해
       - 수평 : 컬럼단위
       - 수직 : 로우단위로 분할

\*\* **성능저하의 원인**

1. 하나의 테이블에 데이터 대량집중 :

-IO문제

-인덱스를 생성할 때 인덱스의 크기(용량)가 커지게 되고 그렇게 되면 인덱스를 찾아가는 단계가 깊어짐

-slect보다는, update에서 성능저하

1. 하나의 테이블에 여러개의 컬럼 : 디스크 점유량 문제, IO 양 문제
2. 대량의 데이터가 처리되는 테이블의 경우 : 인덱스를 적절하게 구성하여 이를 줄일 수 있다.
3. **컬럼**이 많아지는 경우 : 물리적인 **디스크에 여러 블록**에 데이터가 저장되게 된다.

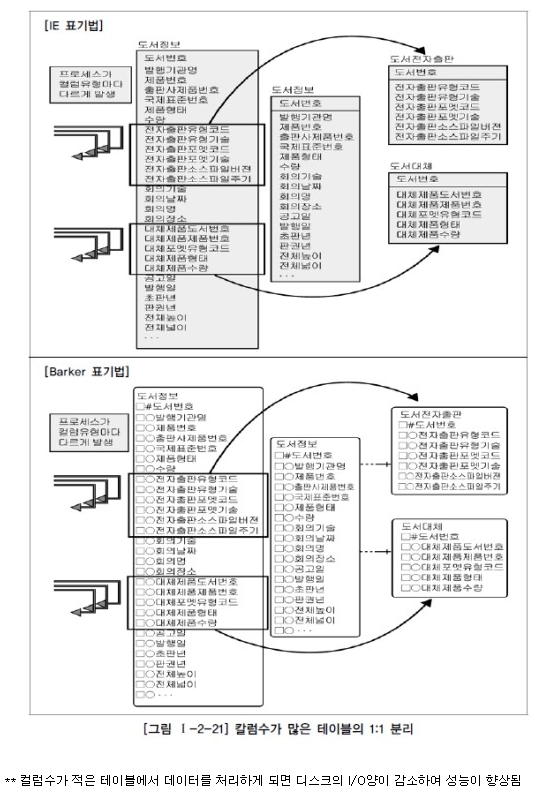
예) 만약 컬럼수가 300개라면?!

* + - * **로우 체이닝 (Row chaining) 현상** : 로우 길이가 너무 길어서 블록 한에 데이터가 모두 저장되지 않고 두개 이상의 블록에 걸쳐 하나의 로우가 저장되어 있는 형태
      * **로우 마이그레이션(Row Migration)** : 데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에서 저장하지 못하고 다른 블록의 빈 공간을 찾아 저장하는 방식

*두 가지 모두 불필요한 I/O가 많이 발생하여 성능이 저하된다.*

1. **한 테이블에 많은 수의 컬럼을 가지고 있는 경우**

*해결책*



1. **대량 데이터 저장 및 처리고 인해 성능이 저하되는 경우**

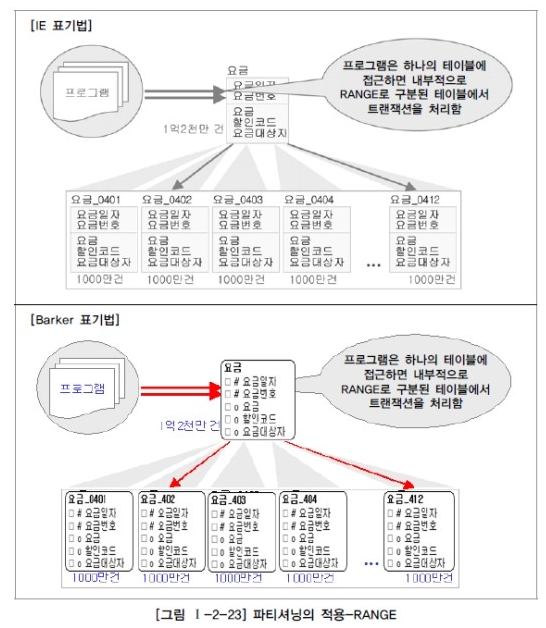
* 위의 경우,
  + **파티셔닝**
  + **PK에 의한 테이블 분할**을 통해 문제를 해결해 볼 수 있다.
    - 보통, 천만건 이상 데이터의 경우, 인덱스를 잘 생성해주더라도 성늠은 좋지 못하다
    - **논리적으로는 하나의 테이블, 물리적으로는 여러 개의 테이블 스페이스에 쪼개어 저장 \_ *파티셔닝***

**\*\* 파티셔닝**

1. Range Partition (가장많이사용)

- **적용대상** : 대상 테이블이 **날짜 or 숫자값으로 분리**가 가능하고, **각 영역별 트랜젝션이 분리**

- **장점** : **데이터 보관주기**에 따라 테이블에 데이터를 쉽게 지우는 것이 가능함

****

1. **List Partition**

장점 : 대용량 데이터를 **특정 값에 따라 분리/저장 가능**

단점 : **쉽게 삭제되는 기능은 없다.**

1. **Hash Partition :** 저장된 해쉬 조건에 따라 해슁 알고리즘이 적용되어 테이블이 분리된다. 데이터의 확인이 어렵고 삭제가 불가능.

**테이블에 대한 수평분할/수직분할의 절차**

**\* 4가지 원칙**

1) 데이터 모델링을 완성한다.

2) 데이터 베이스 용량선정을 한다. (컬럼 수가 어느정도 양의 수준인지 책정)

3) 대량 데이터가 처리되는 테이블에 대해서 **트랜젝션 처리 패턴을 분석한다.**

4) **컬럼 단위**로 집중화된 처리가 발생하는지, **로우 단위**로 집중화된 처리가 발생하는지 분석하여

    집중화된 단위로 테이블을 분리하는 것을 검토한다.

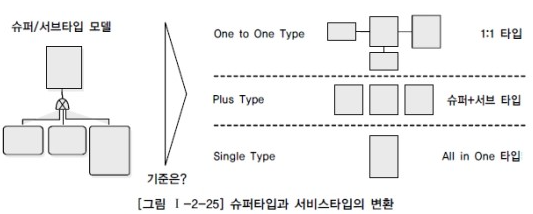
      - 단지 컬럼수가 많다면**1:1로 분리 가능한지 검토**

      - **컬럼수는 적으나 데이터 량이 많다면 ‘파티셔닝’고려**

**데이터 모델과 성능**

**데이터베이스 구조와 성능**

1. **슈퍼/서브타입 모델의 성능고려 방법**
   * 슈퍼타입 : 공통부분
   * 서브타입 : 차이가 있는 속성
   * **논리적 데이터 모델**에서 이용되는 형태이며,  **분석**단계에서 많이 쓰이는 모델
     + One to one type as 1:1
     + Plus type as super + sub type
     + Single type as all in one type

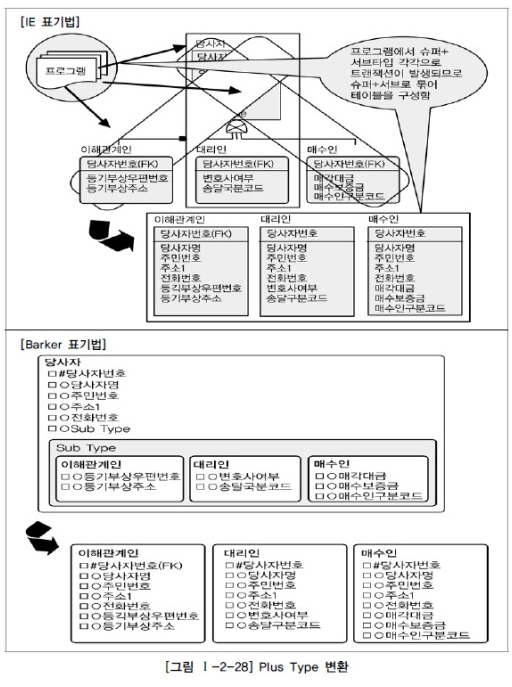


* 성능저하의 원인
  + 트렌젝션은 일괄처리되, 테이블은 개별로 유지되어, union연산에 의해 성능저하
  + 반대로 트렌젝션은 서브타입 개별로 처리, 테이블은 하나로 통합, 불필요하게 많은 양의 데이터가 집약된 이유로 성능이 저하
  + 트랜젝션은 항상 슈퍼+서브 타입을 공통으로 처리하는데 개별로 유지되어 있거나 하나의 테이블로 집약되어 있어 성능이 저하되는 경우

**성능저하의 원인들을 보고도 유추할 수 있듯이, 슈퍼/서브 타입의 변경요인으로 중요한 것은 데이터의 양 및 트렌젝션의 유형이다.**

Example\_1 : 데이터가 소량이라면 데이터처리의 1:1관계를 유지하는 것이 좋음

* **슈퍼/서브 타입의 데이터 모델의 변환 기술**
  + 개별로 발생되는 트랜젝션에 대해서는 개별 테이블로 구성 (One To One)
    - **업무적으로 발생되는 트랜젝션이 슈퍼타입과 서브타입 각각에 대해 발생하는 것**
  + 슈퍼 + 버스타입에 의해 발생되는 트렌젝션에 대해서는 슈퍼 + 서브타입 테이블로 구성
    - **plus type**

****

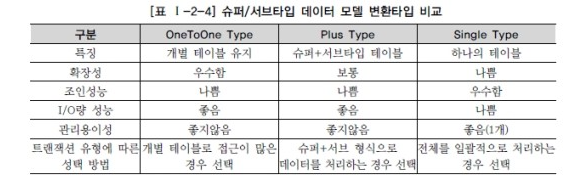
* + 전체를 하나로 묶어 트랜젝션이 발생할 때는 하나의 테이블로 구성

Example\_

만약, 대리인, 매수인, 이해관계인이 항상 한 페이지에 통합 처리될 시

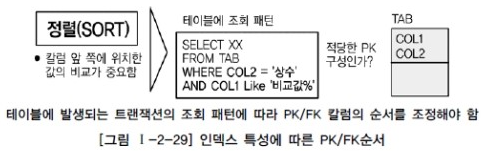
*(반면, Union ALL과 같은 SQL구문이 성능을 저하시킬 수 있다.)*

* **슈퍼/서브타입 데이터 모델의 변환타입 비교**



1. **인덱스 특성을 고려한 PK/FK 데이터베이스 성능향상**

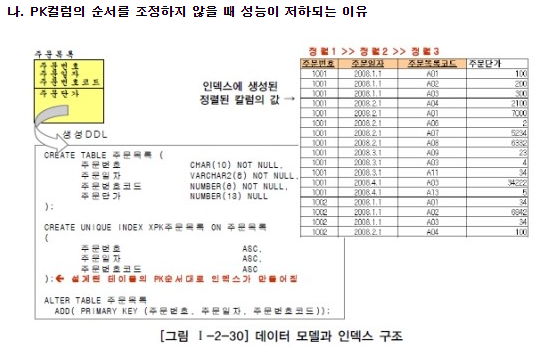
* **PK/FK 컬럼 순서\*\*\*와 성능개요**
  + 인덱스의 중요성 : 데이터를 조작할 때 가장 효과적으로 처리될 수 있도록 접근경로를 제공하는 오브젝트
  + **PK/FK 설계의 중요성** : 데이터에 접근할 때 접근 경로를 제공한다는 측면에서 중요함. 프로젝트시 설계단계 말에 컬럼의 순서를 조정하는 것이 필요하다.



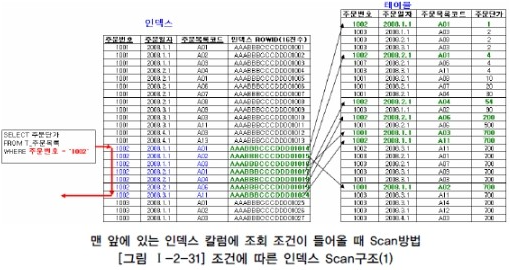
* + PK 순서의 중요성  : 물리적인 데이터 모델링 단계에서는 스스로 생성된 PK 순서 이외에 **다른 엔티티로부터 상속받아 발생되는 PK 순서까지 항상 주의**하여 표시되도록 해야 한다.

​

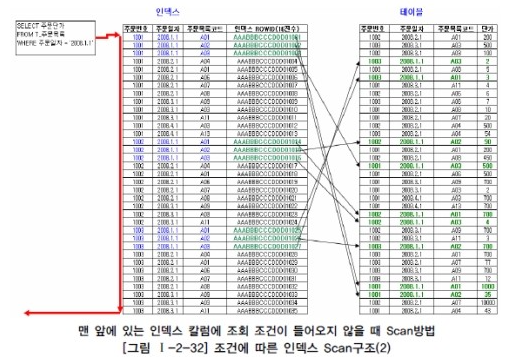
* + FK의 중요성 : FK도 데이터를 조회할 때 **조인의 경로를 제공하는 역할을 수행하므로 이에 대해 반드시 인덱스를 생성하도록 한다.** 조회의 조건도 고려



**: 조회조건에 따라 인덱스를 처리하는 범위가 달라지게 됨**

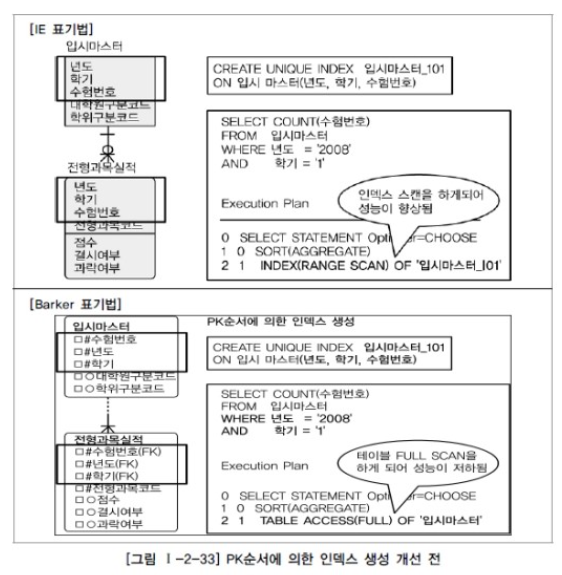
****

만약, 맨 앞에 있는 컬럼이 제외된 상태에서 데이터를 조회할 경우 데이터를 비교하는 범위가 넓어진다 -> 성능저하

****

PK의 순서를 인덱스 특징에 맞게 생성하지 않고 자동으로 생성하게 되면 테이블에 접근하는 트랜젝션의 특징이 효율적이지 않은 인데스가 생성되어 있으므로 인덱스의 범위를 넓히거나 풀 스캔(Full Scan)을 유발해서 성능이 저하된다.

* **PK 순서를 잘못 지정하여 성능이 저하된 경우 –** 간단한 오류

****