성능 데이터 모델링의 개요

1. 성능 데이터 모델링의 정의

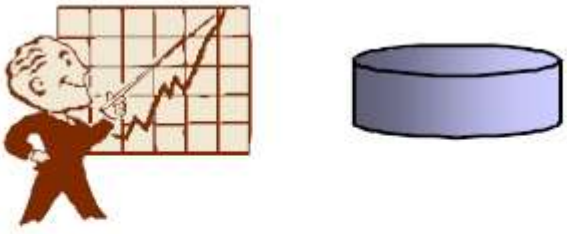
인프라가 갖추어 지지 않은 환경에서 과연 빠른 속도로 이동할 수 있을까? 길이 굽이굽이 굽어져 있고 곳곳에 신호등이 있는 도로에서 아무리 성능이 좋은 차라고 할지라도 과연 그 길을 빠르게 지날 수 있을까? 데이터베이스에서 기본적으로 설계단계에서부터 성능을 고려하지 않고 설계를 하는 것은 빠르게 지나갈 수 없는 길을 지나가는 차에게 빨리 와달라고 요청하는 것과 다름이 없다.

데이터의 용량의 커질수록 기업의 의사결정의 속도가 빨라질수록 데이터를 처리하는 속도는 빠르게 처리되어야 할 필요성을 반증해 준다. 일반적으로 실무 프로젝트에서 보면 잘못된 테이블 디자인 위에서 개발된 애플리케이션의 성능이 저하되는 경우, 개발자가 구축한 SQL 구문에 대해서만 책망을 하는 경우가 많이 있다. 물론 개발자가 SQL 구문을 잘못 구성하여 성능이 저하되는 경우도 있지만 근본적으로 디자인이 잘못되어 SQL 구문을 잘 못 작성하도록 구성될 수밖에 없는 경우도 빈번하게 발생되고 있음을 기억해야 한다.

성능이 저하되는 데이터 모델의 경우 크게 세 가지 경우를 고려하여 그 성능을 향상시킬 수 있다. 데이터 모델 구조에 의해 성능이 저하될 수도 있고 데이터가 대용량이 됨으로 인해 불가피하게 성능이 저하되어 나타나는 경우도 있다. 또한 인덱스 특성을 충분히 고려하지 않고 인덱스를 생성함으로 인해 성능이 저하되어 나타나는 경우도 있다.

일반적으로 성능이라고 하면 데이터조회의 성능을 의미하곤 한다. 그 이유는 데이터입력/수정/삭제는 일시적이고 빈번하지 않고 단건 처리가 많은 반면 데이터조회의 경우는 반복적이고 빈번하며 여러 건을 처리하는 경우가 많기 때문이다. 이러한 특징은 일반적인 트랜잭션의 성격이 조회의 패턴을 가지고 있다는 것이고 업무에 따라서는 입력/수정/삭제의 성능이 중요한 경우도 있다.

따라서 데이터 모델링을 할 때 어떤 작업 유형에 따라 성능 향상을 도모해야 하는지 목표를 분명하게 해야 정확한 성능향상 모델링을 할 수 있음을 기억해야 한다. 성능 데이터 모델링이란 데이터베이스 성능향상을 목적으로 설계단계의 데이터 모델링 때부터 정규화, 반정규화, 테이블통합, 테이블분할, 조인구조, PK, FK 등 여러 가지 성능과 관련된 사항이 데이터 모델링에 반영될 수 있도록 하는 것으로 정의할 수 있다.



성능 데이터 모델링이란 데이터베이스 성능향상을 목적으로 설계단계의 데이터 모델링 때부터 성능과 관련된 사항이 데이터 모델링에 반영될 수 있도록 하는 것이다.

[그림 Ⅰ-2-1] 성능 데이터 모델링

성능 데이터 모델링이 단순히 반정규화만을 의미하지 않음을 주목해야 한다. 성능데이터 모델링은 정규화를 통해서도 수행할 수 있고 인덱스의 특징을 고려해서 칼럼의 순서도 변형할 수 있다. 또한

대량의 데이터특성에 따라 비록 정규화된 모델이라도 테이블을 수직 또는 수평분할하여 적용하는 방법도 있고 논리적인 테이블을 물리적인 테이블로 전환할 때 데이터 처리의 성격에 따라 변환하는 방법도 성능 데이터 모델링의 범주에 포함될 수 있다.

2. 성능 데이터 모델링 수행시점

성능 향상을 위한 비용은 프로젝트 수행 중에 있어서 사전에 할수록 비용이 들지 않는다. 특히 분석/설계 단계에서 데이터 모델에 성능을 고려한 데이터 모델링을 수행할 경우 성능저하에 따른 재업무(Rework) 비용을 최소화 할 수 있는 기회를 가지게 된다. 분석/설계단계에서 데이터 모델은 대충하고, 성능이 저하되는 SQL 문장을 튜닝하고, 부족한 하드웨어 용량(CPU, Memory 등)을 증설하는 등의 작업은 추가적인 비용을 소진하게 하는 원인이 된다. 특히 데이터의 증가가 빠를수록 성능저하에 따른 성능개선비용은 기하급수적으로 증가하게 된다.

[그림 Ⅰ-2-2] 성능 향상 그래프

많은 프로젝트에서 분석/설계단계 때부터 치밀하게 성능에 대비한 설계를 하지 않고, 성능이 저하된 결과만을 대상으로 문제발생 시점에 근시안적인 튜닝을 적용하고 있다. 마치 SQL 튜닝이 모든 것인 것처럼 그것이 마법인 것처럼 SQL 문장에만 집중하여 튜닝을 하는 프로젝트의 현장이 아직도 많이 있다.

따라서 분석/설계 단계에서 데이터베이스 처리 성능을 향상시킬 수 있는 방법을 주도면밀하게 고려해야 한다. 만약 어떤 트랜잭션이 해당 비즈니스 처리에 핵심적이고 사용자 업무처리에 있어 중요함을 가지고 있고 성능이 저하되면 안되는 특징을 가지고 있다면, 프로젝트 초기에 운영환경에 대비한 테스트 환경을 구현하고 그곳에 트랜잭션을 발생시켜 실제 성능을 테스트해 보아야 한다. 이 때 데이터 모델의 구조도 변경하면서 어떠한 구조가 해당 사이트에 성능상 가장 적절한 구조인지를 검토하여 성능이 좋은 모습으로 디자인하는 전략이 요구된다.

3. 성능 데이터 모델링 고려사항

일반적으로 성능 데이터 모델은 다음과 같은 프로세스로 진행하는 것이 데이터 모델링 단계에서 성능을 충분히 고려할 수 있는 방안이 된다.

① 데이터 모델링을 할 때 정규화를 정확하게 수행한다.
② 데이터베이스 용량산정을 수행한다.
③ 데이터베이스에 발생되는 트랜잭션의 유형을 파악한다.
④ 용량과 트랜잭션의 유형에 따라 반정규화를 수행한다.
⑤ 이력모델의 조정, PK/FK 조정, 슈퍼타입/서브타입 조정 등을 수행한다.
⑥ 성능관점에서 데이터 모델을 검증한다.

데이터 모델링을 할 때 기본적으로 정규화를 완벽하게 수행해야 한다. 정규화된 모델이 데이터를 주요 관심사별로 분산시키는 효과가 있기 때문에 그 자체로 성능을 향상시키는 효과가 있다. 일단 정규화가 완성된 모델에 대해서 해당 데이터 모델의 각각의 엔터티에 어느 정도 트랜잭션이 들어오는지 살펴볼 필요가 있다. 이 때 가장 좋은 방법이 엔터티에 대한 용량산정을 하는 것이다. 각각의 엔터티(테이블)에 대한 용량산정을 수행하면 어떤 엔터티(테이블)에 데이터가 집중되는지 파악할 수 있다. 이 용량산정은 엔터티별로 데이터가 대용량인지를 구분하게 하기 때문에 테이블에 대한 성능 고려를 엄격하게 적용해야 하는지 기준이 될 수 있다.

또한 데이터 모델에 발생되는 트랜잭션의 유형을 파악할 필요가 있다. 트랜잭션의 유형에 대한 파악은 CRUD 매트릭스를 보고 파악하는 것도 좋은 방법이 될 수 있고 객체지향 모델링을 적용한다면 시퀀스 다이어그램을 보면 트랜잭션의 유형을 파악하기에 용이하다. 또한 화면에서 처리된 데이터의 종류들을 보면 이벤트(입력, 수정, 삭제, 조회)에 따라 테이블에 데이터가 어떻게 처리되는지를 유추할 수 있다. 트랜잭션의 유형을 파악하게 되면 SQL 문장의 조인관계 테이블에서 데이터조회의 칼럼들을 파악할 수 있게 되어 그에 따라 성능을 고려한 데이터 모델을 설계할 수 있다.

이렇게 파악된 용량산정과 트랜잭션의 유형데이터를 근거로 정확하게 테이블에 대해 반정규화를 적용하도록 한다. 반정규화는 테이블, 속성, 관계에 대해 포괄적인 반정규화의 방법을 적용해야 한다. 또한 대량 데이터가 처리되는 이력모델에 대해 성능고려를 하고 PK/FK 의 순서가 인덱스 특성에 따라 성능에 영향을 미치는 영향도가 크기 때문에 반드시 PK/FK 를 성능이 우수한 순서대로 칼럼의 순서를 조정해야 한다.

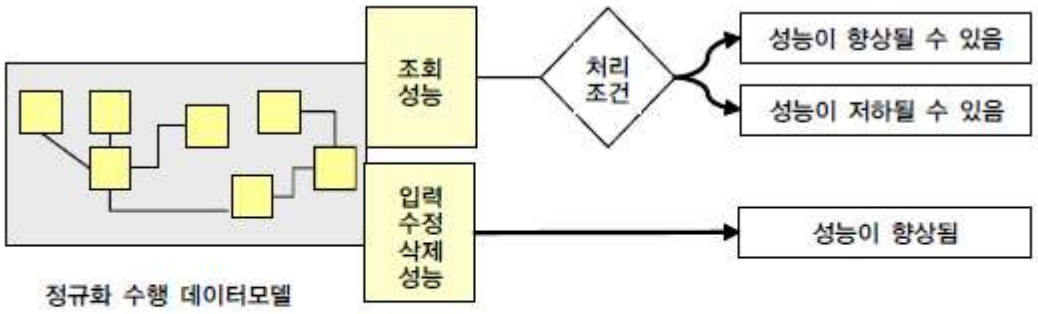
전체적으로 성능에 대한 충분한 고려가 되었는지를 데이터 모델 검토를 통해 다시 한 번 확인하도록 한다. 데이터 모델 검토 시에 일반적인 데이터 모델 규칙만을 검증하지 말고 충분하게 성능이 고려되었는지를 체크리스트에 포함하여 검증하도록 한다.

02. 정규화와 성능 1. 정규화를 통한 성능 향상 전략

데이터 모델링을 하면서 정규화를 하는 것은 기본적으로 데이터에 대한 중복성을 제거하여 주고 데이터가 관심사별로 처리되는 경우가 많기 때문에 성능이 향상되는 특징을 가지고 있다. 물론 엔터티가 계속 발생되므로 SQL 문장에서 조인이 많이 발생하여 이로 인한 성능저하가 나타나는 경우도 있지만 이런 부분은 사례별로 유의하여 반정규화를 적용하는 전략이 필요하다.

정규화를 수행하면 항상 조회 성능이 저하되어 나타날까?
데이터처리의 성능이 무엇인지 정확히 구분하여 인식할 필요가 있다. 데이터베이스에서 데이터를 처리할 때 성능이라고 하면 조회 성능과 입력/수정/삭제 성능의 두 부류로 구분된다. 이 두 가지 성능이 모두 우수하면 좋겠지만 데이터 모델을 구성하는 방식에 따라 두 성능이 Trade-Off 되어 나타나는 경우가 많이 있다.

정규화를 수행한다는 것은 데이터를 결정하는 결정자에 의해 함수적 종속을 가지고 있는 일반속성을 의존자로 하여 입력/수정/삭제 이상을 제거하는 것이다. 데이터의 중복속성을 제거하고 결정자에 의해 동일한 의미의 일반속성이 하나의 테이블로 집약되므로 한 테이블의 데이터 용량이 최소화되는 효과가 있다. 따라서 정규화된 테이블은 데이터를 처리할 때 속도가 빨라질 수도 있고 느려질 수도 있는 특성이 있다.



일반적으로 정규화를 수행해야 데이터처리의 성능이 향상되며
데이터의 조회처리 트랜잭션시에 성능저하가 나타날 수 있음

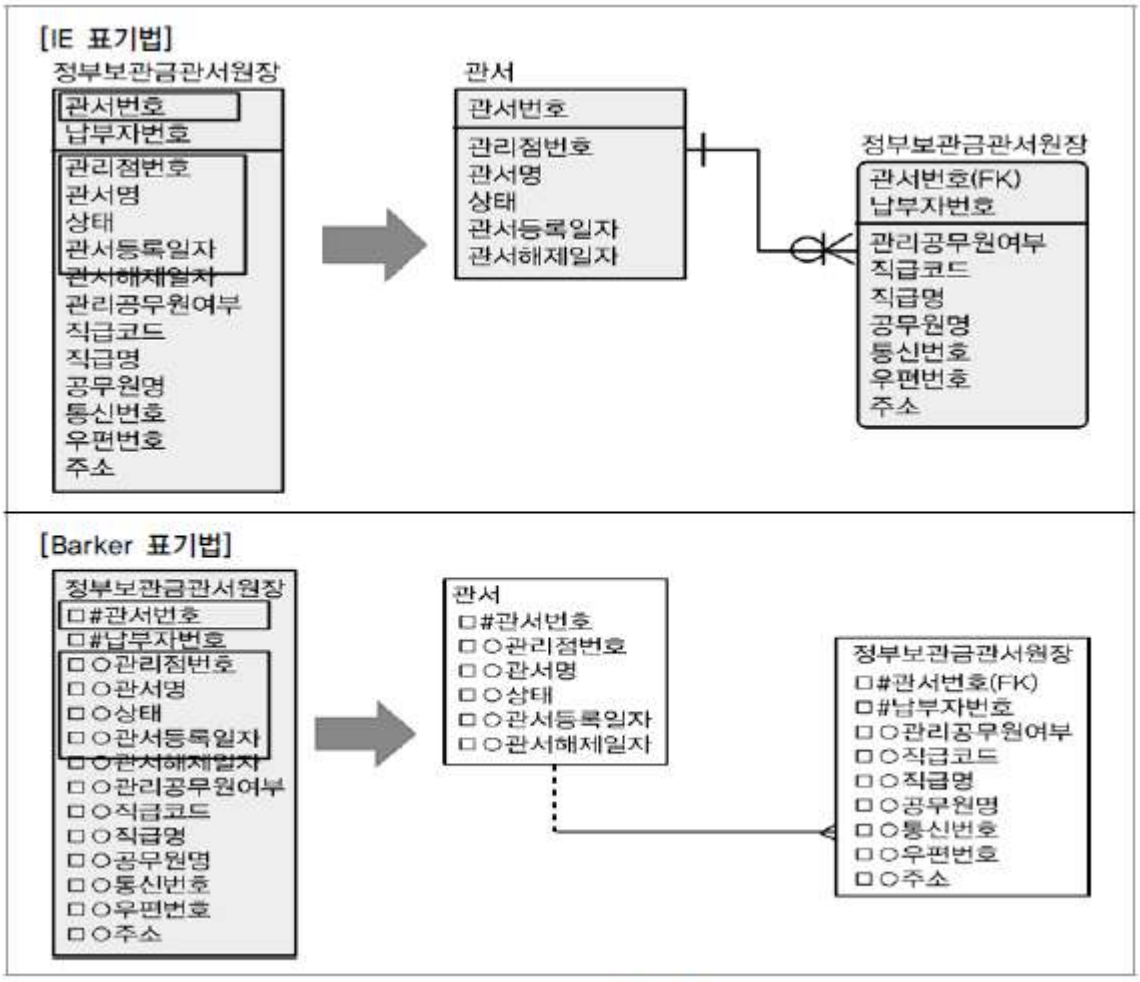
[그림 Ⅰ-2-3] 정규화 수행과 성능

[그림 Ⅰ-2-3]을 보면 정규화 수행 모델은 데이터를 입력/수정/삭제할 때 일반적으로 반정규화된 테이블에 비해 처리 성능이 향상된다. 단 데이터를 조회할 때에는 처리 조건에 따라 조회 성능이 향상될 수도 있고 저하될 수도 있다.

따라서 일반적으로 정규화가 잘 되어 있으면 입력/수정/삭제의 성능이 향상되고 반정규화를 많이 하면 조회의 성능이 향상된다고 인식될 수 있다. 그러나 데이터 모델링을 할 때 반정규화만이 조회 성능을 향상시킨다는 고정관념은 탈피되어야 한다. 정규화를 해서 성능이 저하되기는커녕 정규화를 해야만 성능이 향상되는 경우가 아주 많이 나타나기 때문이다.

2. 반정규화된 테이블의 성능저하 사례 1

정규화하여 조인이 발생하면 성능이 심각하게 저하되는가? 다음 예를 살펴보면 2 차 정규화를 적용한 테이블에 대해서 조인을 하더라도 PK Unique Index 를 이용하면 조인 성능 저하는 미미하게 발생된다.



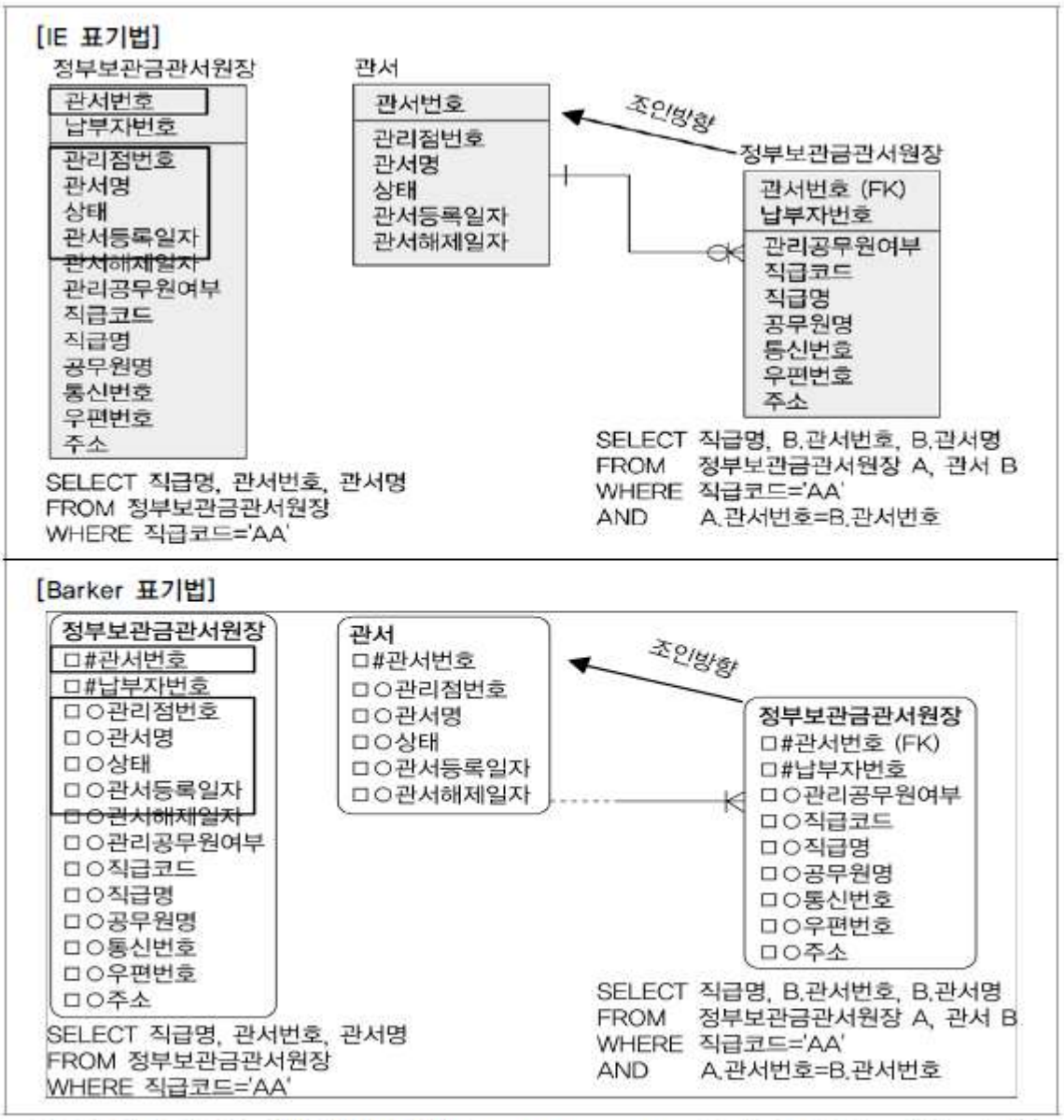
[그림 Ⅰ-2-4] 정규화

[그림 Ⅰ-2-4]의 왼쪽 그림은 2 차 정규화가 안 된 반정규화된 테이블의 모습이고 오른쪽 그림은 부분키 종속을 정규화하여 두 개의 테이블로 분리해 2 차 정규화된 테이블의 모습이다.

2 차 정규화가 안 된 테이블은 직급명과 함께 반정규화된 관서번호, 관서명을 조회하면 하나의 테이블에서 데이터가 조회가 된다. 2 차 정규화된 테이블은 관서번호, 관서명이 관서테이블에만 존재하기 때문에 두 개의 테이블을 조인하여 처리해야 한다.

정부보관금관서원장에서 데이터를 조회하는 것이나, 관서와 정부보관금관서원장을 조인하여 데이터를 조회하나 처리 성능은 사용자가 느끼기에는 거의 차이가 나지 않는다. PK 가 걸려있는 방향으로 조인이 걸려 Unique Index 를 곧바로 찾아서 데이터를 조회하기 때문에, 하나의 테이블에서 조회하는 작업과 비교했을 때 미미하게 성능 차이가 날 뿐 사용자에게 크게 영향을 줄 만큼 성능이 저하되는 일은 없는 것이다.

게다가 위의 예를 '관서등록일자가 2010 년 이후 관서를 모두 조회하라' 는 SQL 구문을 처리하는 것으로 바꾸면, 2 차 정규화된 테이블이 훨씬 빠르다. [그림 Ⅰ-2-5]에서와 같이 왼쪽 테이블에서는 불필요하게 납부자번호만큼 누적된 데이터를 읽어서 결과를 구분하여 보여주어야 하지만 오른쪽은 관서수만큼만 존재하는 데이터를 읽어 곧바로 결과를 보여주기 때문이다. 이렇게 단순한 예만 보아도 정규화를 수행하면 무조건 조회성능이 저하된다는 고정관념이 틀렸다는 것을 알 수 있다.

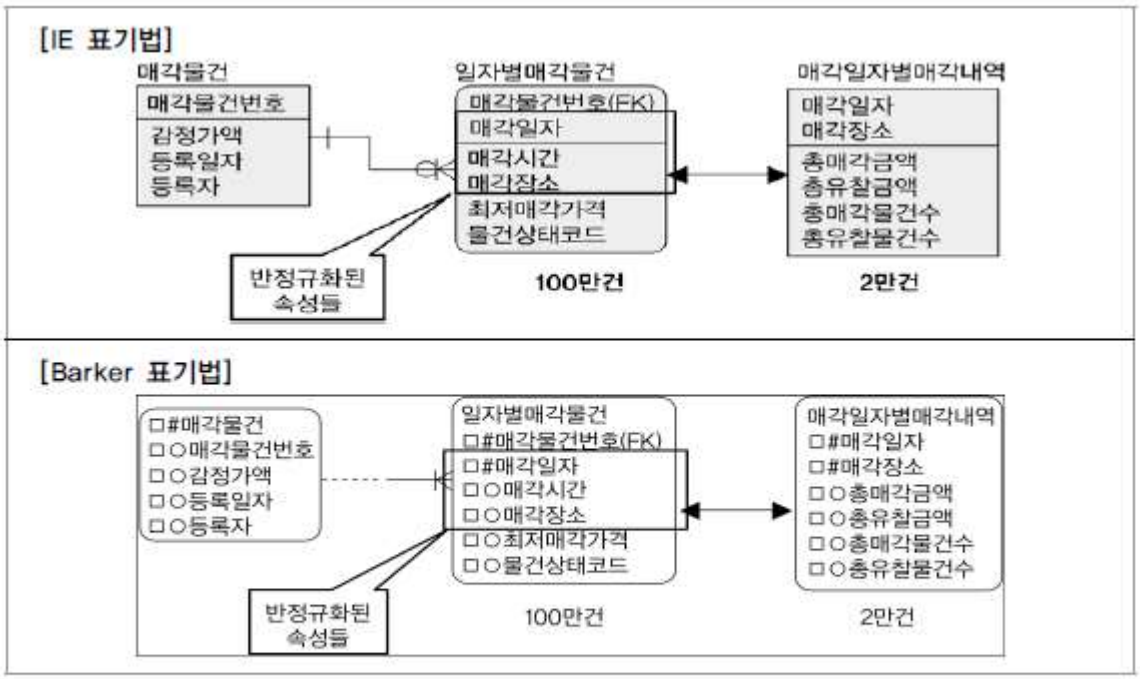


2차 정규화를 적용한 테이블에서 조회해도 PK Unique Index를 이용 조인성능저하는 미미함

[그림 Ⅰ-2-5] 정규화와 SQL

3. 반정규화된 테이블의 성능저하 사례2

두 개의 엔터티가 통합되어 반정규화된 또 다른 경우를 살펴본다. 이 업무에서는 어떤 물건을 매각할 때 매각일자를 정하고 그 일자에 해당하는 매각시간과 매각장소가 결정하는 속성의 성격을 가지고 있다. 즉 매각일자가 결정자가 되고 매각시간과 매각장소가 의존자가 되는 함수적 종속관계가 형성되는 관계이다. 매각일자는 5천 건이 있고 일자별매각물건은 100만 건이 있는 것으로 가정하자.



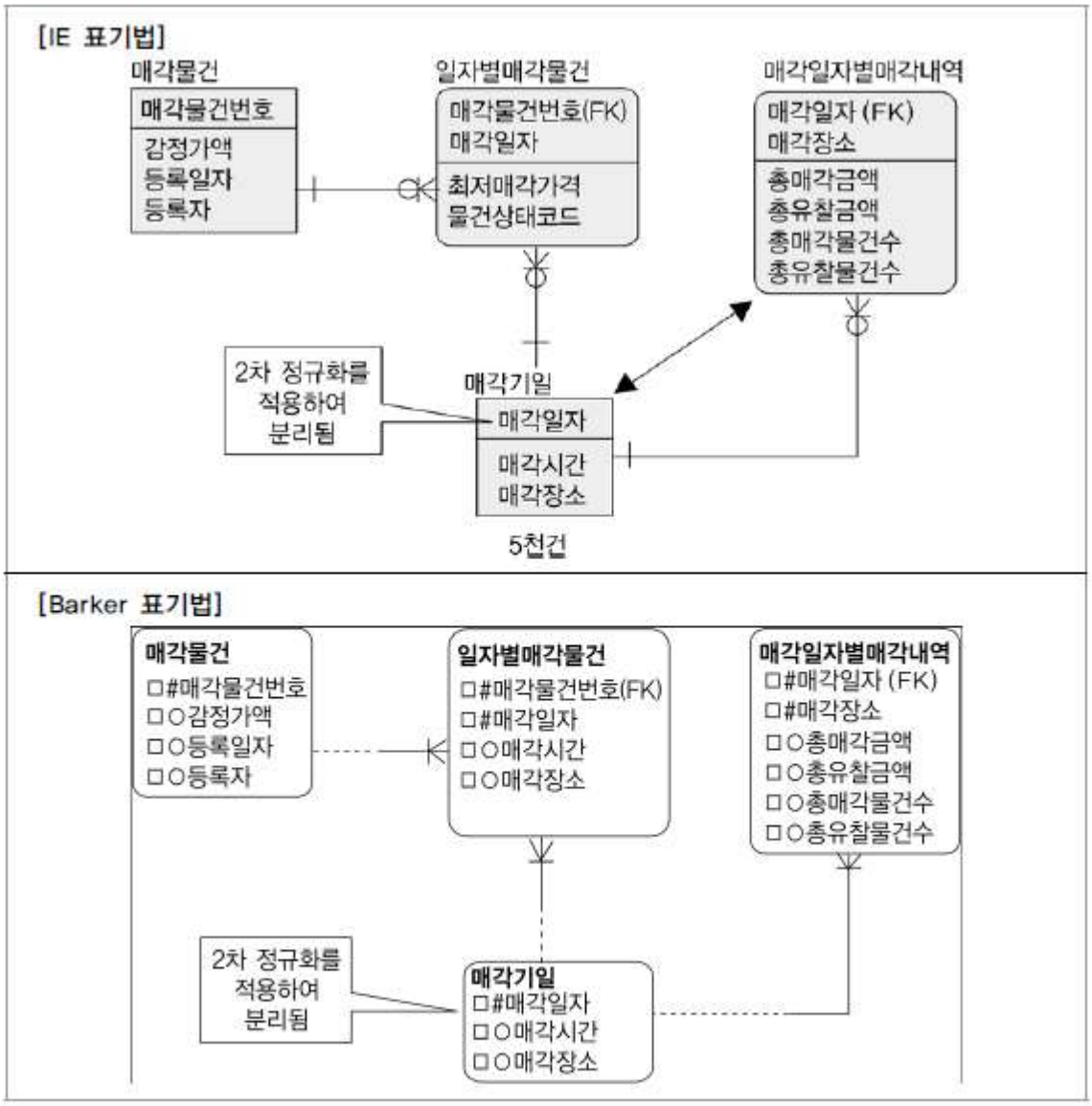
특정 매각장소에 대해 매각일자를 찾아 매각내역을 조회하려면
100만 건의 데이터를 읽어 매각일자를 DISTINCT하여 매각일자별매각내역과 조인이 된다.

[그림 Ⅰ-2-6] 반정규화된 데이터 모델

예를 들어 [그림 Ⅰ-2-6]의 모델에서 '서울 7호'에서 매각된 총매각금액, 총유찰금액을 산출하는 조회용 SQL 문장을 작성하면 다음과 같다.

```
SELECT B.총매각금액 , B.총유찰금액 FROM (SELECT DISTINCT 매각일자 FROM 일자별매각물건 WHERE 매각장소 = '서울 7호') A, <== 100만건의 데이터를 읽어 DISTINCT함 매각일자별매각내역 B WHERE A.매각일자 = B.매각일자 AND A.매각장소 = B.매각장소;
```

대량의 데이터에서 조인 조건이 되는 대상을 찾기 위해 인라인뷰를 사용하기 때문에 성능이 저하된다. 이를 정규화하려면 복합식별자 중에서 일반속성이 주식별자 속성 중 일부에만 종속관계를 가지고 있으므로 2차 정규화의 대상이 된다. 2차 정규화를 적용하면 [그림 Ⅰ-2-7]과 같은 모델이 된다.



특정 매각장소에 대해 매각일자를 찾아 매각내역을 조회하려면 500건의 매각기일과 매각일자별매각내역과 조인이 된다.

[그림 I-2-7] 정규화된 데이터 모델

2 차 정규화를 적용하여 매각일자를 PK 로 하고 매각시간과 매각장소는 일반속성이 되었다. 정규화를 적용했기 때문에 매각일자를 PK 로 사용하는 매각일자별매각내역과도 관계가 연결된다. 따라서 업무흐름에 따른 적확한 데이터 모델링 표기도 가능해지고, 드라이빙이 된 테이블 이 5 천 건의 매각기일 테이블이 되므로 성능도 향상된다.

만약 위의 모델에서 '서울 7 호' 에서 매각된 총매각금액, 총유찰금액을 산출하는 조회용 SQL 문장을 작성하면 다음과 같이 나온다.

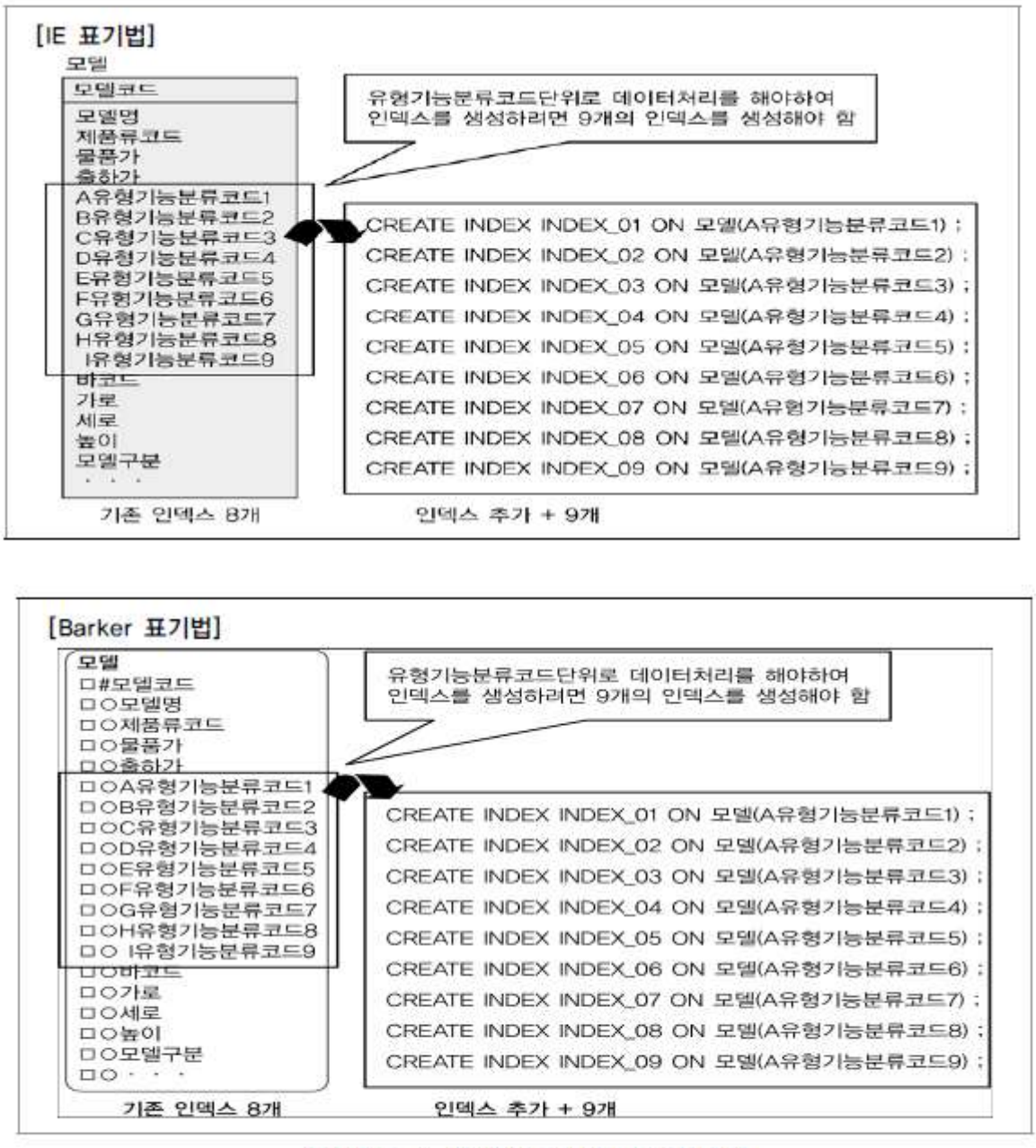
```
SELECT B.총매각금액 , B.총유찰금액 FROM 매각기일 A, 매각일자별매각내역 B WHERE A.매각장소 = '서울7호' <== 5천건의 데이터를 읽음 AND A.매각일자 = B.매각일자 AND A.매각장소 = B.매각장소;
```

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

4. 반정규화된 테이블의 성능저하 사례 3

다음 사례는 동일한 속성 형식을 두 개 이상의 속성으로 나열하여 반정규화한 경우에 해당한다. 계층형 데이터베이스를 많이 사용했던 과거 데이터 모델링의 습관이 남아서인지 관계형 데이터베이스에서도 동일한 속성을 한 테이블에 속성1, 속성2, 속성3 데이터 모델링을 하는 경우가 많이 있다.

[그림 Ⅰ-2-8]을 보면, 모델이라고 하는 테이블에 업무적으로 필요한 8개의 인덱스가 이미 생성되어 있다. 데이터는 30만 건이고 온라인 환경의 데이터베이스라고 가정하자. 유형기능분류코드에 따라 데이터를 조회하는 경우가 많이 나타나 인덱스를 생성하려면 유형기능분류코드 각각에 대해 인덱스를 생성해야 하므로 9개나 되는 인덱스를 추가 생성해야 한다.



[그림 Ⅰ-2-8] 반정규화된 데이터 모델

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. 그런데 [그림 Ⅰ-2-8]의 모델은 다른 필요한 인덱스 이외에 유형기능분류코드 속성에 해당하는 인덱스를 9개나 추가적으로 생성해야 하므로 실전프로젝트에서는 어쩔 수 없이 인덱스를 생성하지 않거나 A유형기능분류코드1 하나만 인덱스를 생성하는 경우가 생긴다. 이에 따라

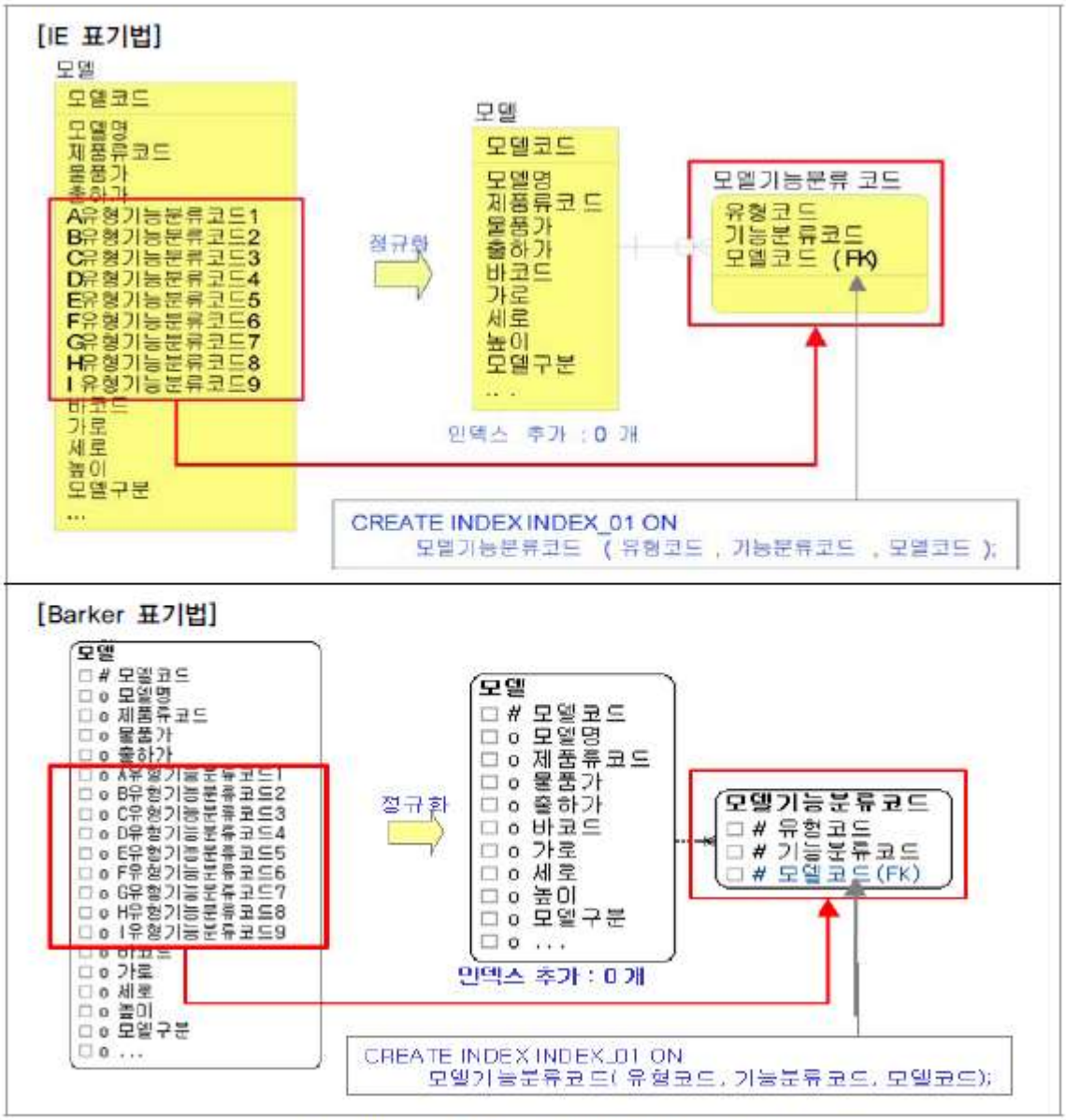
A 유형기능분류코드 1, A 유형기능분류코드 2, A 유형기능분류코드 3...을 이용하면 SQL 의 성능이 저하되는 경우가 많다.

만약 각 유형코드별로 조건을 부여하여 모델코드와 모델명을 조회하는 SQL 문장을 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 ( I유형기능분류코드9 = '09' )
```

각 유형별로 모두 인덱스가 걸려 있어야 인덱스에 의해 데이터를 찾을 수 있다. 이런 모델은 다음과 같이 정규화를 적용해야 한다.

중복속성에 대한 분리가 1 차 정규화의 정의임을 고려하면 모델 테이블은 1 차 정규화의 대상이 된다. 로우단위의 대상도 1 차 정규화의 대상이 되지만 칼럼 단위로 중복이 되는 경우도 1 차 정규화의 대상이 된다. 따라서 모델에 대해 1 차 정규화를 적용하면 [그림 Ⅰ-2-9]와 같이 분리될 수 있다.



[그림 Ⅰ-2-9] 정규화된 데이터 모델

하나의 테이블에 9 개가 반복적으로 나열이 되어 있을 때는 인덱스 생성이 어려웠지만 정규화되어 분리된 이후에는 인덱스 추가 생성이 0 개가 되었다. 또한 분리된 테이블 모델기능분류코드에서 PK 인덱스를 생성하여 이용함으로써 성능이 향상될 수 있다.

만약 각 유형코드별로 조건을 부여하여 모델코드와 모델명을 조회하는 SQL 문장을 작성한다면 다음과 같이 작성된다.

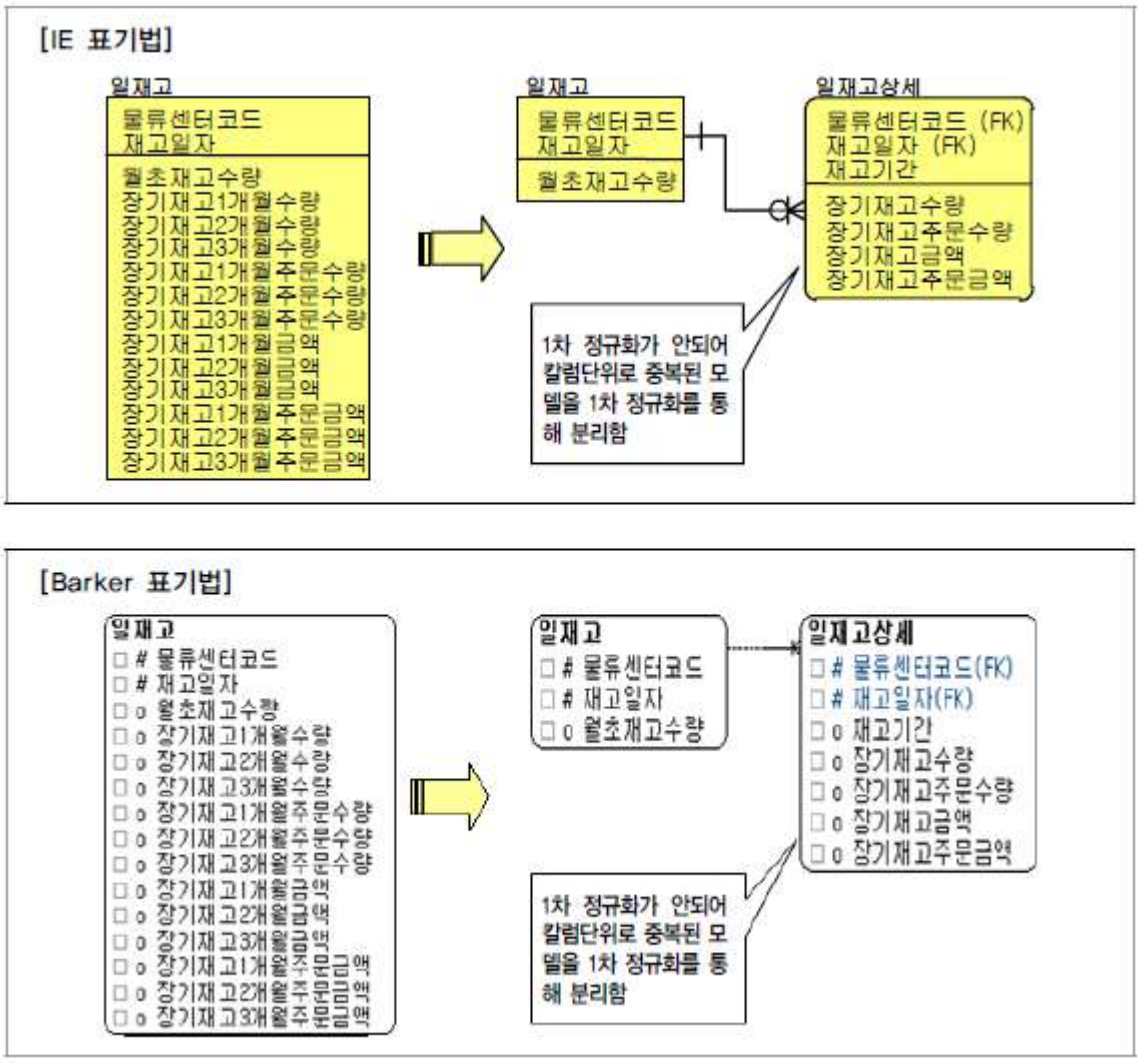
```
SELECT A.모델코드, A.모델명 FROM 모델 A, 모델기능분류코드 B WHERE ( B.유형코드 = 'A' AND B.기능분류코드 = '01' AND A.모델코드 = B.모델코드 ) OR ( B.유형코드 = 'B' AND B.기능분류코드 = '02' AND A.모델코드 = B.모델코드 ) OR ( B.유형코드 = 'C' AND B.기능분류코드 = '07' AND A.모델코드 = B.모델코드 ) OR ( B.유형코드 = 'D' AND B.기능분류코드 = '01' AND A.모델코드 = B.모델코드 ) OR ( B.유형코드 = '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 )
```

위 SQL 구문은 유형코드+기능분류코드+모델코드에 인덱스가 걸려 있으므로 인덱스를 통해 데이터를 조회함으로써 성능이 향상된다.

실전 프로젝트에서도 많은 데이터 모델이 칼럼 단위에서 중복된 경우가 발견된다. 이에 대한 파급효과 계산 없이 무조건 칼럼 단위로 COL1, COL2, COL3... 이런 식으로 데이터 모델링을 하는 것은 근본적으로 SQL 문장의 성능을 나쁘게 하는 결과를 초래할 수도 있으므로 인덱스 생성 영향도를 파악한 이후에 적용하는 것이 좋은 방법이 된다

5. 반정규화된 테이블의 성능저하 사례 4

[그림 Ⅰ-2-10]과 같은 경우도 동일한 사례이다.

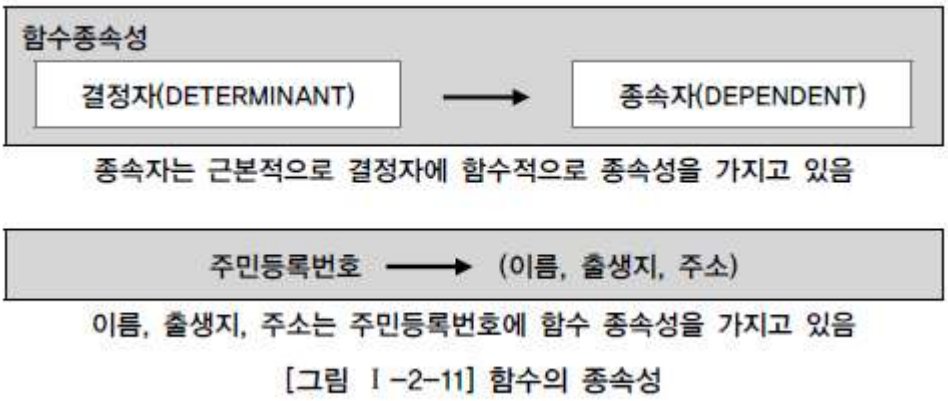


[그림 Ⅰ-2-10] 반정규화된 테이블의 정규화

일재고와 일재고 상세를 구분함으로써 일재고에 발생되는 트랜잭션의 성능저하를 예방할 수 있게 되었다.

6. 함수적 종속성(Functional Dependency)에 근거한 정규화 수행 필요

함수의 종속성(Functional Dependency)은 데이터들이 어떤 기준값에 의해 종속되는 현상을 지칭하는 것이다. 이 때 기준값을 결정자(Determinant)라 하고 종속되는 값을 종속자(Dependent)라고 한다.



[그림 Ⅰ-2-11] 함수의 종속성

[그림 Ⅰ-2-11]에서 보면 사람이라는 엔터티는 주민등록번호, 이름, 출생지, 호주라는 속성이 존재한다. 여기에서 이름, 출생지, 호주라는 속성은 주민등록번호 속성에 종속된다. 만약 어떤 사람의 주민등록번호가 신고되면 그 사람의 이름, 출생지, 호주가 생성되어 단지 하나의 값만을 가지게 된다. 이를 기호로 표시하면, 다음과 같이 표현할 수 있다.

```
주민등록번호 -> (이름, 출생지, 호주)
```

즉 "주민등록번호가 이름, 출생지, 호주를 함수적으로 결정한다."라고 말할 수 있다. 실세계의 데이터들은 대부분 이러한 함수 종속성을 가지고 있다.

함수의 종속성은 데이터가 가지고 있는 근본적인 속성으로 인식되고 있다. 정규화의 궁극적인 목적은 반복적인 데이터를 분리하고 각 데이터가 종속된 테이블에 적절하게(프로세스에 의해 데이터의 정합성이 지켜질 수 있어야 함) 배치되도록 하는 것이므로 이 함수의 종속성을 이용하여 정규화 작업이나 각 오브젝트에 속성을 배치하는 작업에 이용이 되는 것이다.

기본적으로 데이터는 속성간의 함수종속성에 근거하여 정규화되어야 한다. 프로젝트 수행에서 정규화는 선택사항이 아니라 필수사항이다. 데이터의 구조를 함수적 종속관계에 의해 정규화 사상에 맞게 분리하는 것은 마치 건축물을 지을 때 부실공사를 방지하기 위해 하중, 균형, 내구성을 고려한 기본적인 설계를 하는 것과 같다. 정규화 이론은 건축으로 따지면 기본적인 설계 원칙만큼이나 중요한 이론이다.

프로젝트에서 정규화가 적용된 모델을 설계하기 위해서는 정규화에 대한 기본이론을 정확하고 구체적으로 이해하고 있어야 한다. 전문적인 IT 프로젝트를 진행하는 설계자가 정규화의 이론을 모르고 데이터 모델링을 설계하는 것은 불가능하다. 눈이 어두운 사람이 감각에 의해 길을 찾아가는 것과 같다. 분명하게 자기가 가는 길을 명확하게 검증 받을 수 있도록, 지도를 가지고 길을 가듯이 정규화의 이론을 숙지하고 정보시스템의 근간인 데이터를 설계해야 한다.

03. 반정규화와 성능

1. 반정규화를 통한 성능향상 전략

가. 반정규화의 정의

반정규화(=역정규화) 용어는 조금 다르게 표현되어도 그 의미는 동일하다. 여기에서 반정규화는 '반(Half)'의 의미가 아닌 한자로 반대하다의 의미를 가진 '反'의 의미이다. 영어로는 De-Normalization 이다. 비정규화는 아예 정규화를 수행하지 않은 모델을 지칭할 때 사용한다.

반정규화를 정의하면 정규화된 엔터티, 속성, 관계에 대해 시스템의 성능향상과 개발(Development)과 운영(Maintenance)의 단순화를 위해 중복, 통합, 분리 등을 수행하는 데이터 모델링의 기법을 의미한다. 협의의 반정규화는 데이터를 중복하여 성능을 향상시키기 위한 기법이라고 정의할 수 있고 좀 더 넓은 의미의 반정규화는 성능을 향상시키기 위해 정규화된 데이터 모델에서 중복, 통합, 분리 등을 수행하는 모든 과정을 의미한다.

데이터 무결성이 깨질 수 있는 위험을 무릅쓰고 데이터를 중복하여 반정규화를 적용하는 이유는 데이터를 조회할 때 디스크 I/O 량이 많아서 성능이 저하되거나 경로가 너무 멀어 조인으로 인한 성능 저하가 예상되거나 칼럼을 계산하여 읽을 때 성능이 저하될 것이 예상되는 경우 반정규화를 수행하게 된다.

중복성의 원리를 활용하여 데이터조회시 성능을 향상시키는 역할을 할 수 있음

[그림 I-2-12] 반정규화

기본적으로 정규화는 입력/수정/삭제에 대한 성능을 향상시킬 뿐만 아니라 조회에 대해서도 성능을 향상시키는 역할을 한다. 그러나 정규화만을 수행하면 엔터티의 갯수가 증가하고 관계가 많아져 일부 여러 개의 조인이 걸려야만 데이터를 가져오는 경우가 있다. 이러한 경우 업무적으로 조회에 대한 처리성능이 중요하다고 판단될 때 부분적으로 반정규화를 고려하게 되는 것이다. 또한 정규화의 함수적 종속관계는 위반하지 않지만 데이터의 중복성을 증가시켜야만 데이터조회의 성능을 향상시키는 경우가 있다. 이러한 경우 반정규화를 통해서 성능을 향상시킬 수 있게 되는 것이다.

프로젝트에서는 설계단계에서 반정규화를 적용하게 되는데 반정규화를 기술적으로 수행하지 않는 경우에는 다음과 같은 현상이 발생된다.

- 성능이 저하된 데이터베이스가 생성될 수 있다.
- 구축단계나 시험단계에서 반정규화를 적용할 때 수정에 따른 노력비용이 많이 들게 된다.

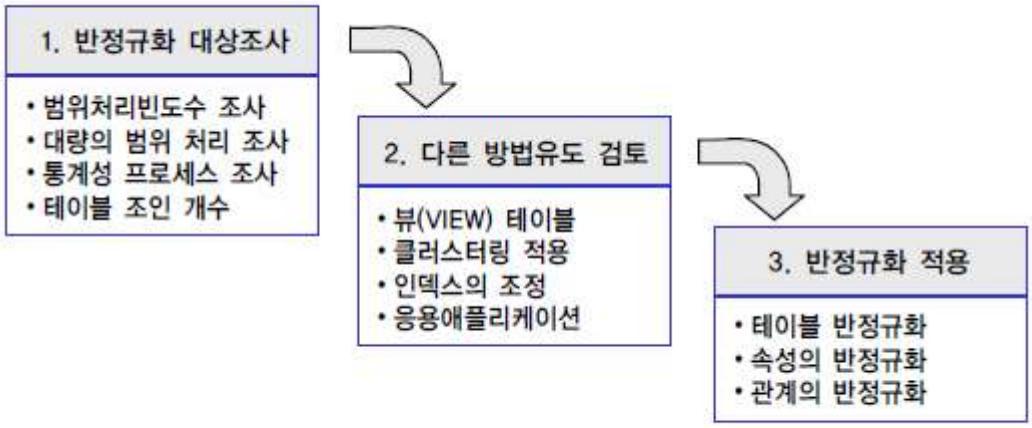
나. 반정규화의 적용방법

반정규화도 하나의 난이도 높은 데이터 모델링의 실무기술이다. 보통 프로젝트에서는 칼럼 중복을 통해서만 반정규화를 수행하게 된다. 칼럼의 반정규화가 많은 이유는 개발을 하다가 SQL 문장 작성이 복잡해지고 그에 따라 SQL 단위 성능 저하가 예상이 되어 다른 테이블에서 조인하여 가져와야

할 칼럼을 기준이 되는 테이블에 중복하여 SQL 문장을 단순하게 처리하도록 하기 위해 요청하는 경우가 많다. 이 때문에 칼럼의 반정규화 유형이 많이 나타나게 된다. 이렇게 무분별하게 칼럼의 반정규화를 많이 하게 되는 것은 데이터에 대한 무결성을 깨뜨리는 결정적인 역할을 하는 경우가 많이 있다.

반정규화에 대한 필요성이 결정이 되면 칼럼의 반정규화 뿐만 아니라 테이블의 반정규화와 관계의 반정규화를 종합적으로 고려하여 적용해야 한다. 또한 반정규화를 막연하게 중복을 유도하는 것만을 수행하기 보다는 성능을 향상시킬 수 있는 다른 방법들을 고려하고 그 이후에 반정규화를 적용하도록 해야 한다.

반정규화를 적용할 때는 기본적으로 데이터 무결성이 깨질 가능성이 많이 있기 때문에 반드시 데이터 무결성을 보장할 수 있는 방법을 고려한 이후에 반정규화를 적용하도록 해야 한다. 정규화와 반정규화 사이에는 Trade-Off 관계 즉, 마치 저울추가 양쪽에 존재하여 한쪽이 무거워지면 다른 쪽은 위로 올라가는 것처럼 정규화만을 강조하다 보면 성능의 이슈가 발생될 수 있고 반정규화를 과도하게 적용하다 보면 데이터 무결성이 깨질 수 있는 위험이 증가하게 되는 것이다. 따라서 반정규화를 적용할 때에는 데이터 무결성이 중요함을 알고 데이터 무결성이 충분히 유지될 수 있도록 프로세스 처리에 있어서 안정성이 먼저 확인이 되어야 한다.



[그림 Ⅰ-2-13] 반정규화 절차

반정규화를 적용하기 위해서는 [그림 Ⅰ-2-13]에 나타난 것처럼 먼저 반정규화의 대상을 조사하고 다른 방법을 적용할 수 있는지 검토하고 그 이후에 반정규화를 적용하도록 한다.

- 반정규화의 대상을 조사한다.

일단 전체 데이터의 양을 조사하고 그 데이터가 해당 프로세스를 처리할 때 성능저하가 나타날 수 있는지 검증해야 한다. 데이터가 대량이고 성능이 저하될 것으로 예상이 되면 다음 4 가지 경우를 고려하여 반정규화를 고려하게 된다.

- 자주 사용되는 테이블에 접근(Access)하는 프로세스의 수가 많고 항상 일정한 범위만을 조회하는 경우에 반정규화를 검토한다.

- 테이블에 대량의 데이터가 있고 대량의 데이터 범위를 자주 처리하는 경우에 처리범위를 일정하게 줄이지 않으면 성능을 보장할 수 없을 경우에 반정규화를 검토한다.

- 통계성 프로세스에 의해 통계 정보를 필요로 할 때 별도의 통계테이블(반정규화 테이블)을 생성한다.

- 테이블에 지나치게 많은 조인(JOIN)이 걸려 데이터를 조회하는 작업이 기술적으로 어려울 경우 반정규화를

검토한다.

- 반정규화의 대상에 대해 다른 방법으로 처리할 수 있는지 검토한다.
가급적이면 데이터를 중복하여 데이터 무결성을 깨뜨릴 위험을 제어하기 위하여 반정규화를 결정하기 이전에 성능을 향상시킬 수 있는 다른 방법을 모색하도록 한다.

- 지나치게 많은 조인(JOIN)이 걸려 데이터를 조회하는 작업이 기술적으로 어려울 경우 뷰(VIEW)를 사용하면 이를 해결할 수도 있다. 뷰가 조회의 성능을 향상시키는 역할을 수행하지는 않는다. 다만 개발자별로 SQL 문장을 만드는 방법에 따라 성능저하가 나타날 수 있으므로 성능을 고려한 뷰를 생성하여 개발자가 뷰를 통해 접근하게 함으로써 성능저하의 위험을 예방하는 것도 좋은 방법이 된다.

- 대량의 데이터처리나 부분처리에 의해 성능이 저하되는 경우에 클러스터링을 적용하거나 인덱스를 조정함으로써 성능을 향상시킬 수 있다. 클러스터링을 적용하는 방법은 대량의 데이터를 특정 클러스터링 팩트에 의해 저장방식을 다르게 하는 방법이다. 이 방법의 경우 데이터를 입력/수정/삭제하는 경우 성능이 많이 저하되므로 조회중심의 테이블이 아니라면 생성하면 안되는 오브젝트이다. 다만, 조회가 대부분이고 인덱스를 통해 성능향상이 불가능하다면 클러스터링을 고려할 만하다. 또한 인덱스를 통해 성능을 충분히 확보할 수 있다면 인덱스를 조정하여 반정규화를 회피하도록 한다.

- 대량의 데이터는 Primary Key 의 성격에 따라 부분적인 테이블로 분리할 수 있다. 즉 파티셔닝 기법(Partitioning)이 적용되어 성능저하를 방지할 수 있다. 인위적인 테이블을 통합/분리하지 않고 물리적인 저장기법에 따라 성능을 향상시킬 수 있는 파티셔닝을 고려해 볼 수 있다. 이 경우는 데이터가 특정 기준(파티셔닝 키)에 의해 다르게 저장되고 파티셔닝 키에 따른 조회가 될 때 성능이 좋아지는 특성이 있다. 따라서 특정 기준에 의해 물리적인 저장공간이 구분될 수 있고 트랜잭션이 들어올 때 일정한 기준에 의해 들어온다면 파티셔닝 테이블을 적용하여 조회의 성능을 향상시키는 것도 좋은 방법이 될 수 있다.

- 응용 애플리케이션에서 로직을 구사하는 방법을 변경함으로써 성능을 향상시킬 수 있다. 응용 메모리 영역에 데이터를 처리하기 위한 값을 캐쉬한다든지 중간 클래스 영역에 데이터를 캐쉬하여 공유하게 하여 성능을 향상 시키는 것도 성능을 향상시키는 방법이 될 수 있다.

- 반정규화를 적용한다.
반정규화를 적용하기 이전에 사전에 충분히 성능에 대한 고려가 이루어져서 반정규화를 적용해야겠다는 판단이 들었다면 이 때 반정규화의 세 가지 규칙을 고려하여 반정규화를 적용하도록 한다. 반정규화를 하는 대상으로는 테이블, 속성, 관계에 대해 적용할 수 있으며 꼭 테이블과 속성, 관계에 대해 중복으로 가져가는 방법만이 반정규화가 아니고 테이블, 속성, 관계를 추가할 수도 있고 분할할 수도 있으며 제거할 수도 있다.

성능을 향상시킬 수 있는 포괄적인 방법을 적용하여 반정규화를 적용하는 것이 전문화된 반정규화의 기법임을 기억할 필요가 있다.

2. 반정규화의 기법

넓은 의미에서 반정규화를 고려할 때 성능을 향상시키기 위한 반정규화는 여러 가지가 나타날 수 있다.

가. 테이블 반정규화

[표 Ⅰ-2-1] 테이블의 반정규화

기법분류	기법	내용
테이블병합	1:1 관계 테이블병합	1:1 관계를 통합하여 성능향상
	1:M 관계 테이블병합	1:M 관계 통합하여 성능향상
	슈퍼/서브타입 테이블병합	슈퍼/서브 관계를 통합하여 성능향상
테이블분할	수직분할	칼럼단위의 테이블을 디스크 I/O를 분산처리 하기 위해 테이블을 1:1로 분리하여 성능향상(트랜잭션의 처리되는 유형을 파악이 선행되어야 함)
	수평분할	로우 단위로 집중 발생되는 트랜잭션을 분석하여 디스크 I/O및 데이터접근의 효율성을 높여 성능을 향상하기 위해 로우단위로 테이블을 쪼갬(관계가 없음)
테이블추가	중복테이블 추가	다른 업무이거나 서버가 다른 경우 동일한 테이블구조를 중복하여 원격조인을 제거하여 성능을 향상
	통계테이블 추가	SUM, AVG 등을 미리 수행하여 계산해 둠으로써 조회 시 성능을 향상
	이력테이블 추가	이력테이블 중에서 마스터 테이블에 존재하는 레코드를 중복하여 이력테이블에 존재하는 방법은 반정규화의 유형
	부분테이블 추가	하나의 테이블의 전체 칼럼 중 자주 이용하는데 자주 이용하는 집중화된 칼럼들이 있을 때 디스크 I/O를 줄이기 위해 해당 칼럼들을 모아놓은 별도의 반정규화된 테이블을 생성

나. 칼럼 반정규화

[표 Ⅰ-2-2] 칼럼의 반정규화

반정규화 기법	내용
중복칼럼 추가	조인에 의해 처리할 때 성능저하를 예방하기 위해 즉, 조인을 감소시키기 위해 중복된 칼럼을 위치시킴
파생칼럼 추가	트랜잭션이 처리되는 시점에 계산에 의해 발생되는 성능저하를 예방하기 위해 미리 값을 계산하여 칼럼에 보관함. Derived Column이라고 함
이력테이블 칼럼추가	대량의 이력데이터를 처리할 때 불특정 날 조회나 최근 값을 조회할 때 나타날 수 있는 성능저하를 예방하기 위해 이력테이블에 기능성 칼럼(최근값 여부, 시작과 종료일자 등)을 추가함
PK에 의한 칼럼 추가	복합의미를 갖는 PK를 단일 속성으로 구성하였을 경우 발생됨. 단일 PK안에서 특정 값을 별도로 조회하는 경우 성능저하가 발생될 수 있음. 이 때 이미 PK안에 데이터가 존재하지만 성능향상을 위해 일반속성으로 포함하는 방법이 PK의한 칼럼추가 반정규화임
응용시스템 오작동을 위한 칼럼 추가	업무적으로는 의미가 없지만 사용자가 데이터처리를 하다가 잘못 처리하여 원래 값으로 복구하기를 원하는 경우 이전 데이터를 임시적으로 중복하여 보관하는 기법. 칼럼으로 이것을 보관하는 방법은 오작동 처리를 위한 임시적인 기법이지만 이것을 이력데이터 모델로 풀어내면 정상적인 데이터 모델의 기법이 될 수 있음

다. 관계 반정규화

[표 Ⅰ-2-3] 관계의 반정규화

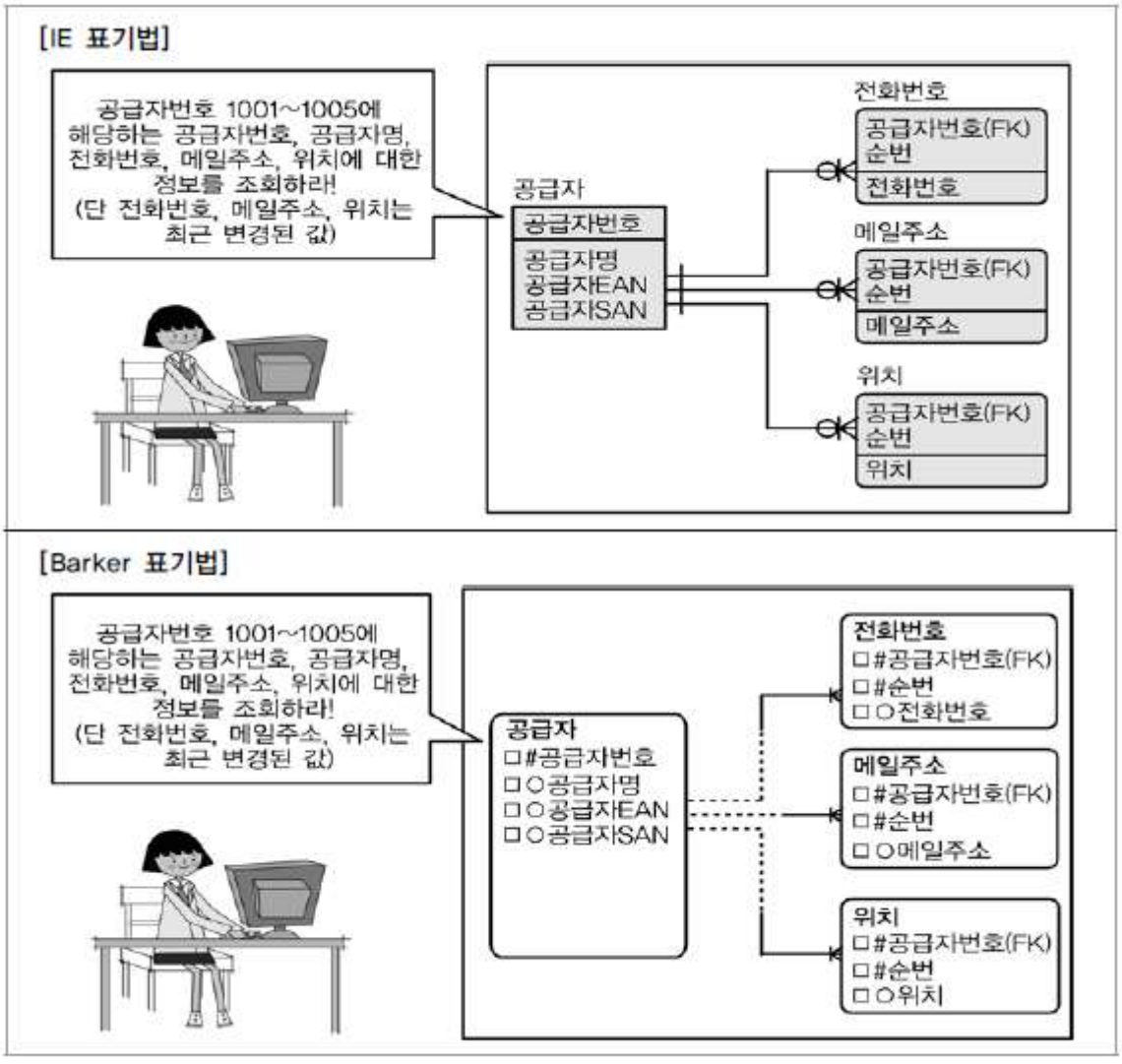
반정규화 기법	내용
중복관계 추가	데이터를 처리하기 위한 여러 경로를 거쳐 조인이 가능하지만 이 때 발생할 수 있는 성능저하를 예방하기 위해 추가적인 관계를 맺는 방법이 관계의 반정규화임

데이블과 칼럼의 반정규화는 데이터 무결성에 영향을 미치게 되나 관계의 반정규화는 데이터 무결성을 깨뜨릴 위험을 갖지 않고서도 데이터처리의 성능을 향상시킬 수 있는 반정규화의 기법이 된다. 데이터 모델 전체가 관계로 연결되어 있고 관계가 서로 먼 친척간에 조인관계가 빈번하게 되어 성능저하가 예상이 된다면 관계의 반정규화를 통해 성능향상을 도모할 필요가 있다

테이블과 칼럼의 반정규화는 데이터 무결성에 영향을 미치게 되나 관계의 반정규화는 데이터 무결성을 깨뜨릴 위험을 갖지 않고서도 데이터처리의 성능을 향상시킬 수 있는 반정규화의 기법이 된다. 데이터 모델 전체가 관계로 연결되어 있고 관계가 서로 먼 친척간에 조인관계가 빈번하게 되어 성능저하가 예상이 된다면 관계의 반정규화를 통해 성능향상을 도모할 필요가 있다.

3. 정규화가 잘 정의된 데이터 모델에서 성능이 저하될 수 있는 경우

[그림 Ⅰ-2-14]는 공급자라고 하는 엔터티가 마스터이고 전화번호와 메일주소 위치가 각각 변경되는 내용이 이력형태로 관리되는 데이터 모델이다. 이 모델에서 공급자정보를 가져오는 경우를 가정해 보자.



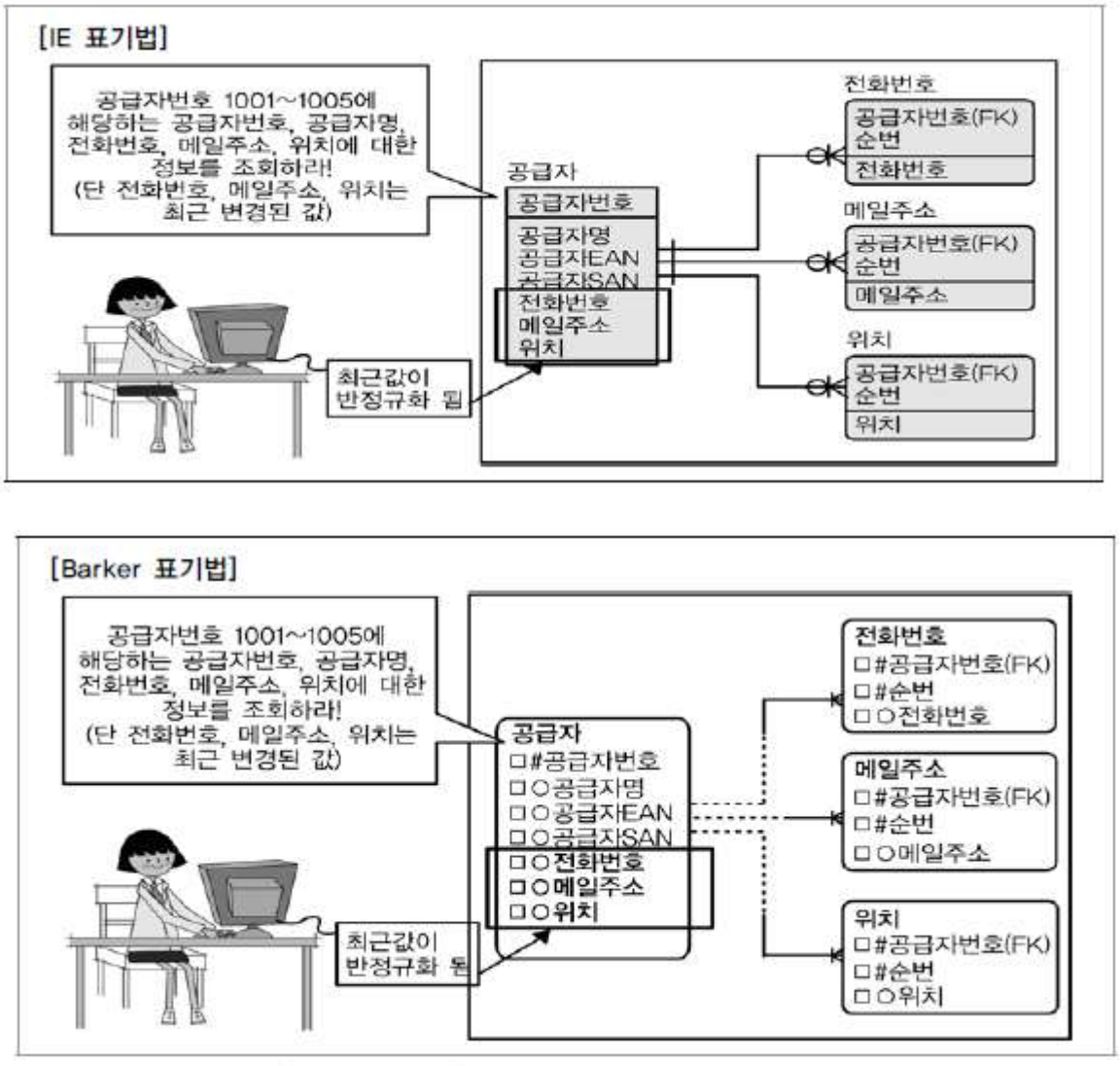
[그림 Ⅰ-2-14] 정규화 모델의 문제점

공급자와 전화번호, 메일주소, 위치는 1:M 관계이므로 한 명의 공급자당 여러 개의 전화번호, 메일주소, 위치가 존재한다. 따라서 가장 최근에 변경된 값을 가져오기 위해서는 조금 복잡한 조인이 발생될 수 밖에 없다.

다음 SQL 은 위와 같은 조건을 만족하는 SQL 구문이 된다.

```
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, (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 SQL 구문은 쉽게 나올 수 있다. 이른바 A4 용지 5 장으로 작성된 SQL 이 쉽지 않게 발견될 수 있는 것이다.
위의 모델을 적절하게 반정규화를 적용하면 즉, 가장 최근에 변경된 값을 마스터에 위치시키면 다음과 같이 아주 간단한 SQL 구문이 작성 된다.



[그림 Ⅰ-2-15] 반정규화를 통한 성능향상 사례

위에서 복잡하게 작성된 SQL 문장이 반정규화를 적용하므로 인해 다음과 같이 간단하게 작성이 되어 가독성도 높아지고 성능도 향상되어 나타났다.

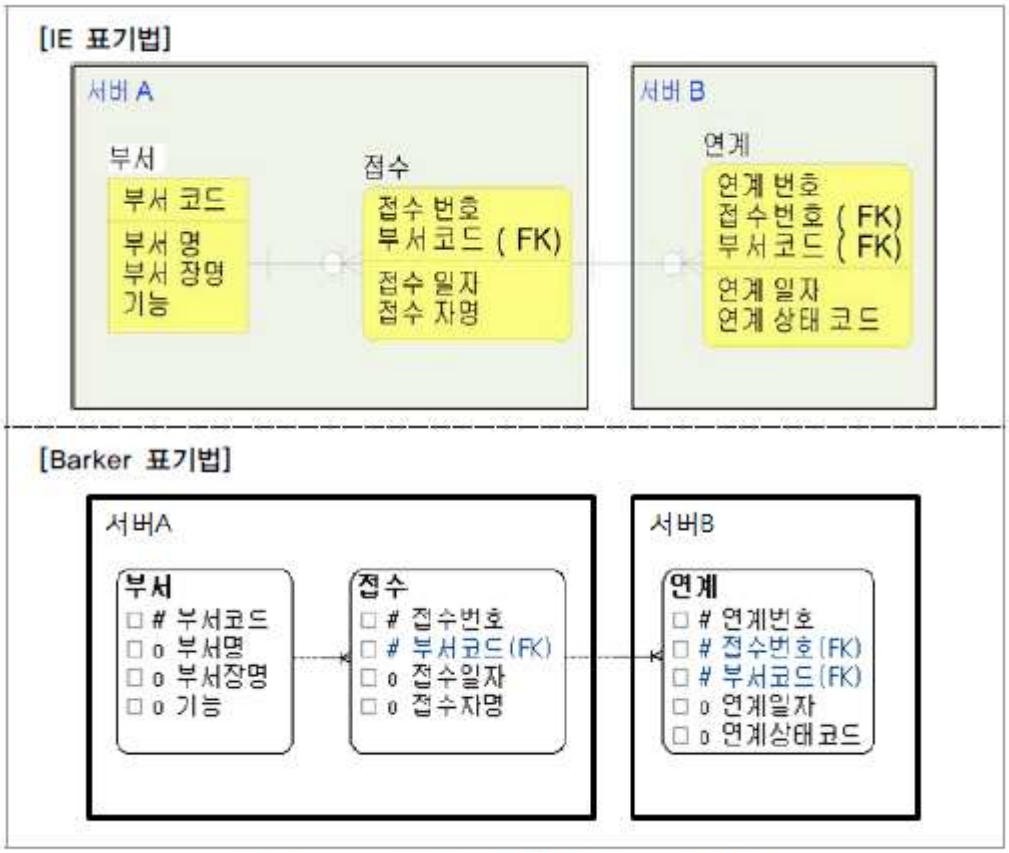
```
SELECT 공급자명, 전화번호, 메일주소, 위치 FROM 공급자 WHERE 공급자번호 BETWEEN '1001' AND '1005'
```

결과만 보면 너무 당연하고 쉬운 것 같지만 기억해야 할 사실은 위 내용들은 모두 실제로 프로젝트를 할 때도 이와 같이 SQL 문장의 성능과 단순성을 고려하지 않고 무모하게 설계되는 경우가 많이 있다는 점이다.

4. 정규화가 잘 정의된 데이터 모델에서 성능이 저하된 경우

업무의 영역이 커지고 다른 업무와 인터페이스가 많아짐에 따라 데이터베이스서버가 여러 대인 경우가 있다. [그림 Ⅰ-2-16]은 데이터베이스서버가 분리 되어 분산데이터베이스가 구성되어 있을 때 반정규화를 통해 성능을 향상시킬 수 있는 경우이다.

서버 A 에 부서와 접수 테이블이 있고 서버 B 에 연계라는 테이블이 있는데 서버 B 에서 데이터를 조회할 때 빈번하게 조회되는 부서번호가 서버 A 에 존재하기 때문에 연계, 접수, 부서 테이블이 모두 조인이 걸리게 된다. 게다가 분산데이터베이스 환경이기 때문에 다른 서버간에도 조인이 걸리게 되어 성능이 저하되는 것이다.

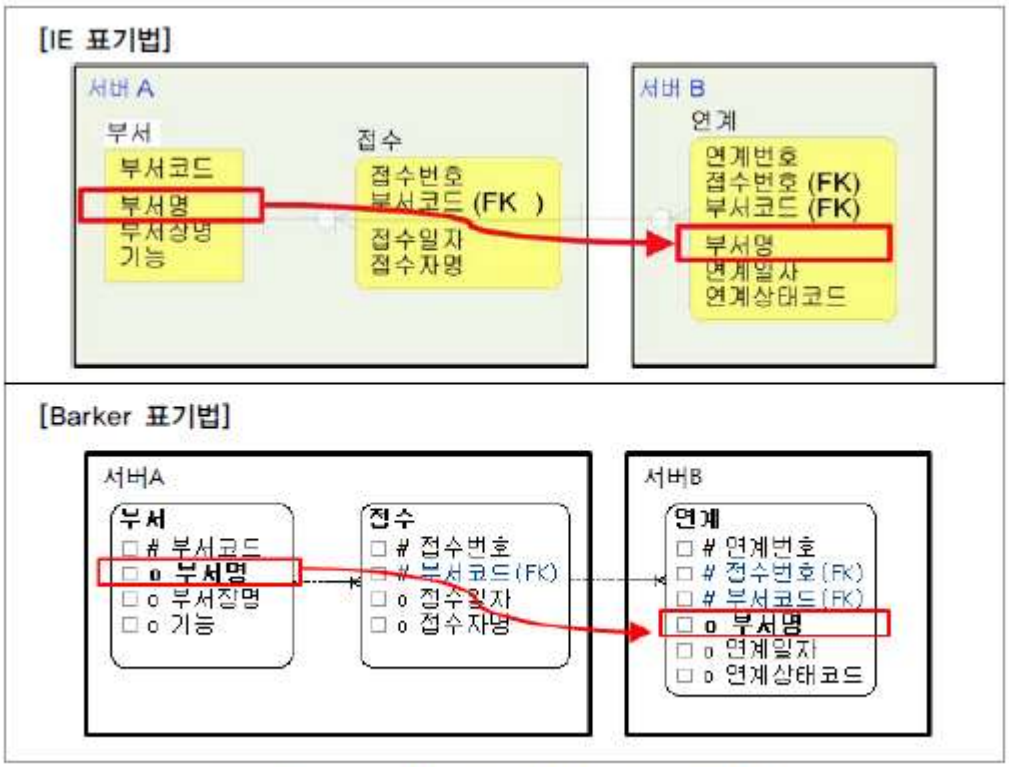


[그림 Ⅰ-2-16] 다른 서버간 정규화된 사례

위의 모델을 통해 서버 B 의 연계테이블에서 부서명에 따른 연계상태코드를 가져오는 SQL 구문은 다음과 같이 작성된다.

```
SELECT C.부서명, A.연계상태코드 FROM 연계 A, 접수 B, 부서 C <== 서버 A 와 서버 B 가 조인이 걸림
WHERE A.부서코드 = B.부서코드 AND A.접수번호 = B.접수번호 AND B.부서코드 = C.부서코드 AND A.연계일자 BETWEEN '20040801' AND '20040901'
```

Oracle 의 경우 DB LINK 조인이 발생하여 일반조인보다 성능이 저하될 것이다.
위의 분산 환경에 따른 데이터 모델을 다음과 같이 서버 A 에 있는 부서테이블의 부서명을 서버 B 의 연계테이블에 부서명으로 속성 반정규화를 함으로써 조회 성능을 향상시킬 수 있다.



[그림 Ⅰ-2-17] 다른 서버간 반정규화된 사례

[그림 Ⅰ-2-17]의 모델에 대한 SQL 구문은 다음과 같이 작성된다.

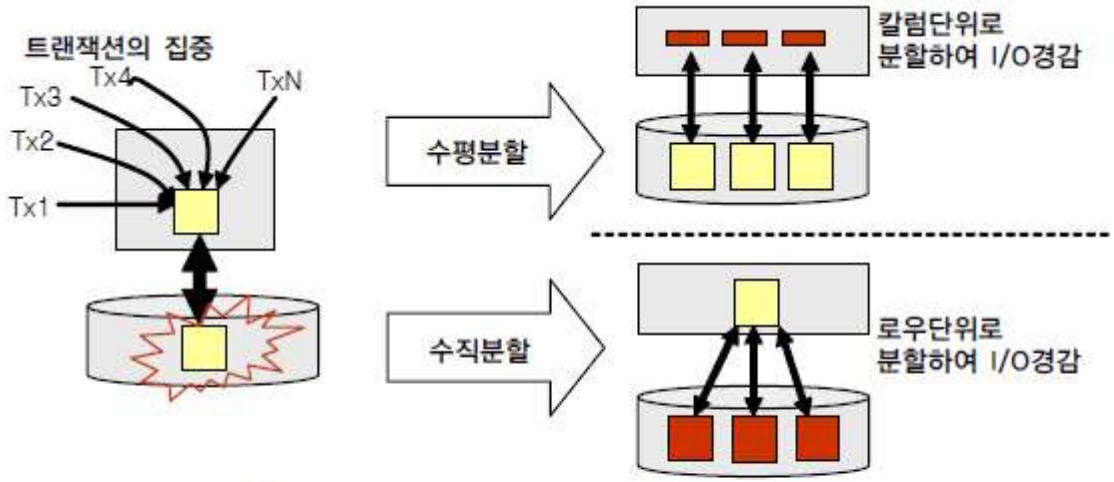
```
SELECT 부서명, 연계상태코드 FROM 연계 WHERE 연계일자 BETWEEN '20040801' AND '20040901'
```

SQL 구문도 간단해지고 분산되어 있는 서버간에도 DB LINK 조인이 발생하지 않아 성능이 개선되었다.
반정규화를 적용할 때 기억해야 할 내용은 데이터를 입력, 수정, 삭제할 때는 성능이 떨어지는 점을 기억해야 하고 데이터의 무결성 유지에 주의를 해야 한다.

04.대량 데이터에 따른 성능

1. 대량 데이터발생에 따른 테이블 분할 개요

아무리 설계가 잘되어 있는 데이터 모델이라고 하더라도 대량의 데이터가 하나의 테이블에 집약되어 있고 하나의 하드웨어 공간에 저장되어 있으면 성능저하를 피하기가 힘들다. 이런 원리는 하나의 고속도로 차선을 넓게 시공하여 건설해도 교통량이 많게 되면 이 넓은 도로가 정체현상을 보이는 것과 비슷한 원리로 이해할 수 있다. 일의 처리되는 양이 한군데에 몰리는 현상은 어떤 업무에 있어서 중요한 업무에 해당되는 데이터가 특정 테이블에 있는 경우에 발생이 되는데 이런 경우 트랜잭션이 분산 처리될 수 있도록 테이블단위에서 분할의 방법을 적용할 필요가 있는 것이다.



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

한 테이블에 데이터가 대량으로 집중되거나 하나의 테이블에 여러 개의 칼럼이 존재하여 디스크에 많은 블록을 점유하는 경우는 모두 성능저하를 유발할 수 있는 경우이다. 하나의 테이블에 대량의 데이터가 존재하는 경우에는 인덱스의 Tree 구조가 너무 커져 효율성이 떨어져 데이터를 처리(입력, 수정, 삭제, 조회)할 때 디스크 I/O 를 많이 유발하게 된다. 또한 한 테이블에 많은 수의 칼럼이 존재하게 되면 데이터가 디스크의 여러 블록에 존재하므로 인해 디스크에서 데이터를 읽는 I/O 량이 많아지게 되어 성능이 저하되게 된다.

대량의 데이터가 처리되는 테이블에 성능이 저하되는 이유는 SQL 문장에서 데이터를 처리하기 위한 I/O 의 양이 증가하기 때문이다. 당연하게 데이터의 양이 많아지면 그것을 처리하기 위한 I/O 량이 많아질 것이라고 막연하게 생각할 수 있지만, 인덱스를 적절하게 구성하여 이용하게 하면 I/O 를 줄일 수 있을 것이라고 생각할 수 있다. 조회조건에 따른 인덱스를 적절하게 이용하면 해당 테이블에 데이터가 아무리 많아도 원하는 데이터만 접근하면 되기 때문에 I/O 의 양이 그다지 증가하지 않을 것으로 생각할 수 있다. 그러나 대량의 데이터가 하나의 테이블에 존재하게 되면 인덱스를 생성할 때 인덱스의 크기(용량)가 커지게 되고 그렇게 되면 인덱스를 찾아가는 단계가 깊어지게 되어 조회의 성능에도 영향을 미치게 된다. 인덱스 크기가 커질 경우 조회의 성능에는 영향을 미치는 정도가 작지만 데이터를 입력/수정/삭제하는 트랜잭션의 경우 인덱스의 특성상 일량이 증가하여 더 많이 성능의 저하를 유발하게 된다. 또한 데이터에 대한 범위 조회시 더 많은 I/O 유발할 수 있게 되어 성능저하를 유발할 수 있게 된다.

칼럼이 많아지게 되면 물리적인 디스크에 여러 블록에 데이터가 저장되게 된다. 따라서 데이터를 처리할 때 여러 블록에서 데이터를 I/O 해야 하는 즉 SQL 문장의 성능이 저하될 수 특징을 가지게 된다. 물론, 테이블에 칼럼이 많아지는 현상은 정규화이론인 함수적 종속성에 근거하여 당연히 하나의 테이블에 설계할 수는 있다. 그러나 대량 데이터를 가진 테이블에서 불필요하게 많은 양의 I/O 를 유발하여 성능이 저하되는 경우에는 이것을 기술적으로 분석하여 성능을 향상하는 방법으로 분할할 수 있다.

[그림 I-2-19] 디스크에 데이터저장의 개념

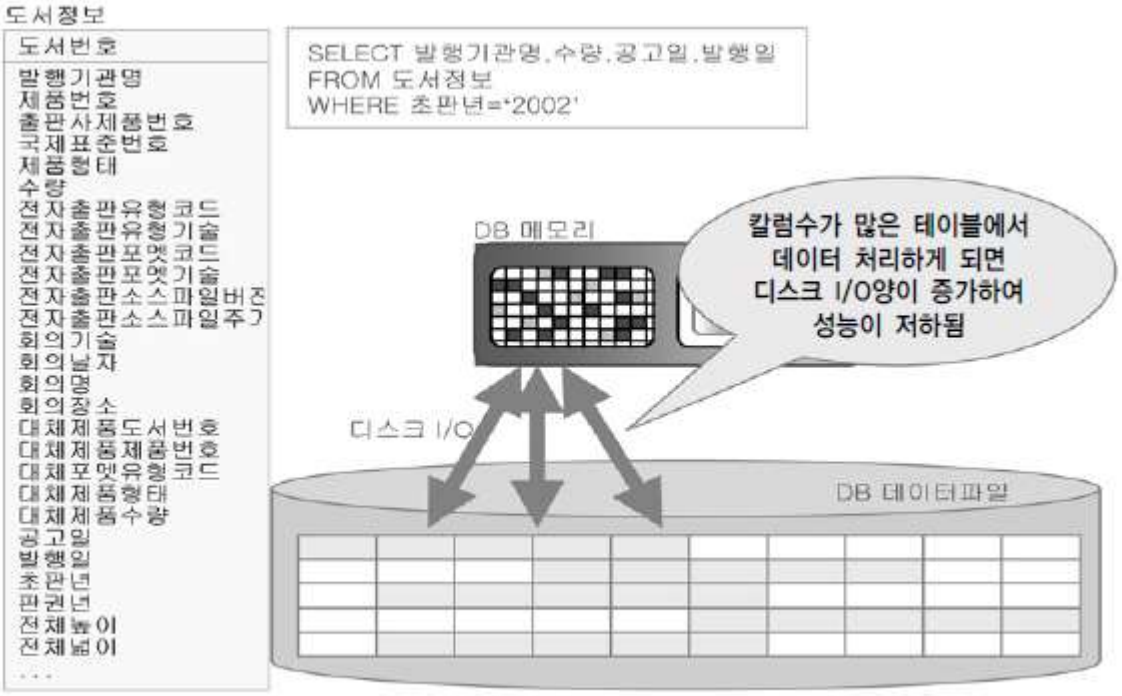
프로젝트를 수행할 때 때로는 하나의 테이블에 300 개 이상의 칼럼을 가지고 있는 경우가 있다. 컴퓨터 화면 하나에는 볼 수가 없어서 스크롤을 하면서 하나의 테이블에 있는 칼럼을 구경해야 할 정도이다. 이렇게 많은 칼럼은 로우체이닝과 로우마이그레이션이 많아지게 되어 성능이 저하된다.

로우 길이가 너무 길어서 데이터 블록 하나에 데이터가 모두 저장되지 않고 두 개 이상의 블록에 걸쳐 하나의 로우가 저장되어 있는 형태가 로우체이닝(Row Chaining) 현상이다. 또한 로우마이그레이션(Row Migration)은 데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에서 저장하지 못하고 다른 블록의 빈 공간을 찾아 저장하는 방식이다. 로우체이닝과 로우마이그레이션이 발생하여 많은 블록에 데이터가 저장되면 데이터베이스 메모리에서 디스크와 I/O(입력/출력)가 발생할 때 불필요하게 I/O 가 많이 발생하여 성능이 저하된다.

2. 한 테이블에 많은 수의 칼럼을 가지고 있는 경우

도서정보라고 하는 테이블에 칼럼수가 아주 많은 경우를 생각해 보자. 생략된 칼럼까지 합하면 대략 200 개라고 가정한다. 만약 하나의 로우의 길이가 10KByte 라고 하고 블록은 2K 단위로 쪼개져 있다고 가정한다. 또한 블록에 데이터는 모두 채워진다고 가정하면 대략 하나의 로우는 5 블록에 걸쳐 데이터가 저장될 것이다.

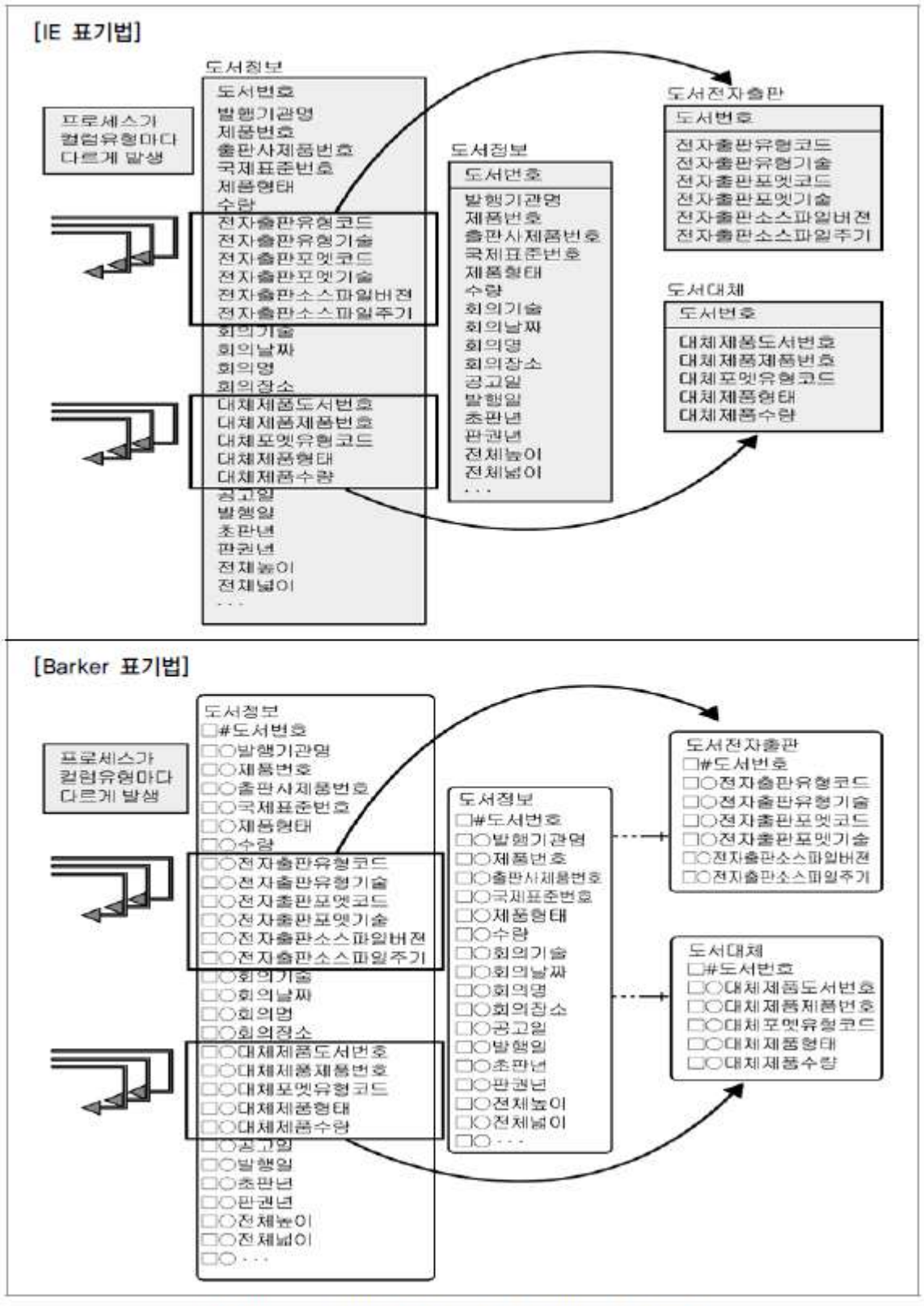
이 때 칼럼의 앞쪽에 위치한 발행기관명, 수량, 중간에 위치한 공고일, 발행일에 대한 정보를 가져오려면 물리적으로 칼럼의 값이 블록에 넓게 산재되어 있어 디스크 I/O 가 많이 일어나게 된다.



[그림 Ⅰ-2-20] 칼럼수가 많은 테이블

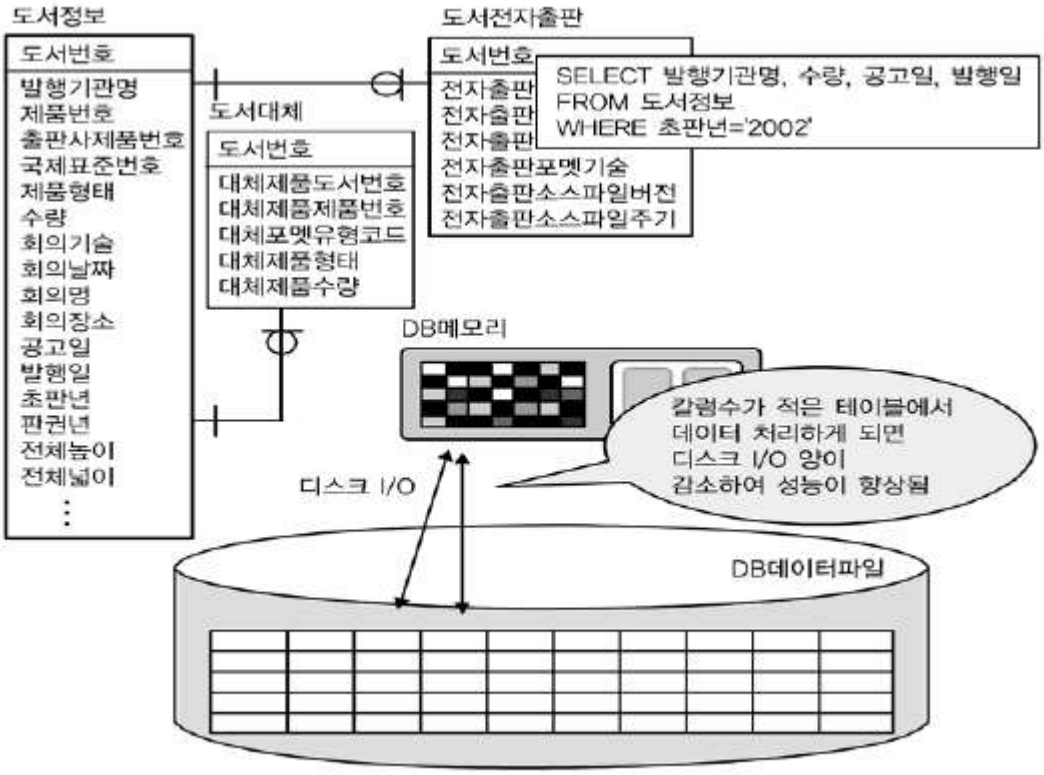
200 개의 칼럼을 동시에 조회하여 화면에 보여주는 경우는 드문 사례이다. 만약 200 개의 칼럼이 가지고 있는 값을 모두 한 화면에 보여주기 위해서는 화면을 몇 번 스크롤 하면서 보여야 한다. 즉 이렇게 많은 칼럼을 가지고 있는 테이블에 대해서는 트랜잭션이 발생될 때 어떤 칼럼에 대해 집중적으로 발생하는지 분석하여 테이블을 쪼개어 주면 디스크 I/O 가 감소하게 되어 성능이 개선되게 된다.

[그림 Ⅰ-2-21]의 데이터 모델을 살펴보자.



[그림 Ⅰ-2-21] 칼럼수가 많은 테이블의 1:1 분리

도서정보 테이블에는 전자출판유형에 대한 트랜잭션이 독립적으로 발생이 되는 경우가 많고 대체제품에 대한 유형의 트랜잭션이 독립적으로 발생되는 경우가 많이 있어 1:1 관계로 분리하였다. 분리된 테이블은 디스크에 적어진 칼럼이 저장이 되므로 로우마이그레이션과 로우체이닝이 많이 줄어들 수 있다. 따라서 아래와 같이 발행기관명, 수량, 중간에 위치한 공고일, 발행일을 가져오는 동일한 SQL 구문에 대해서도 디스크 I/O 가 줄어들어 성능이 개선되게 된다.



[그림 I-2-22] 칼럼수가 많은 테이블의 1:1 분리의 I/O

많은 수의 칼럼을 가지는 데이터 모델 형식도 실전 프로젝트에서 흔히 나타나는 현상이다. 트랜잭션을 분석하여 적절하게 1:1 관계로 분리함으로써 성능향상이 가능하도록 해야 한다.

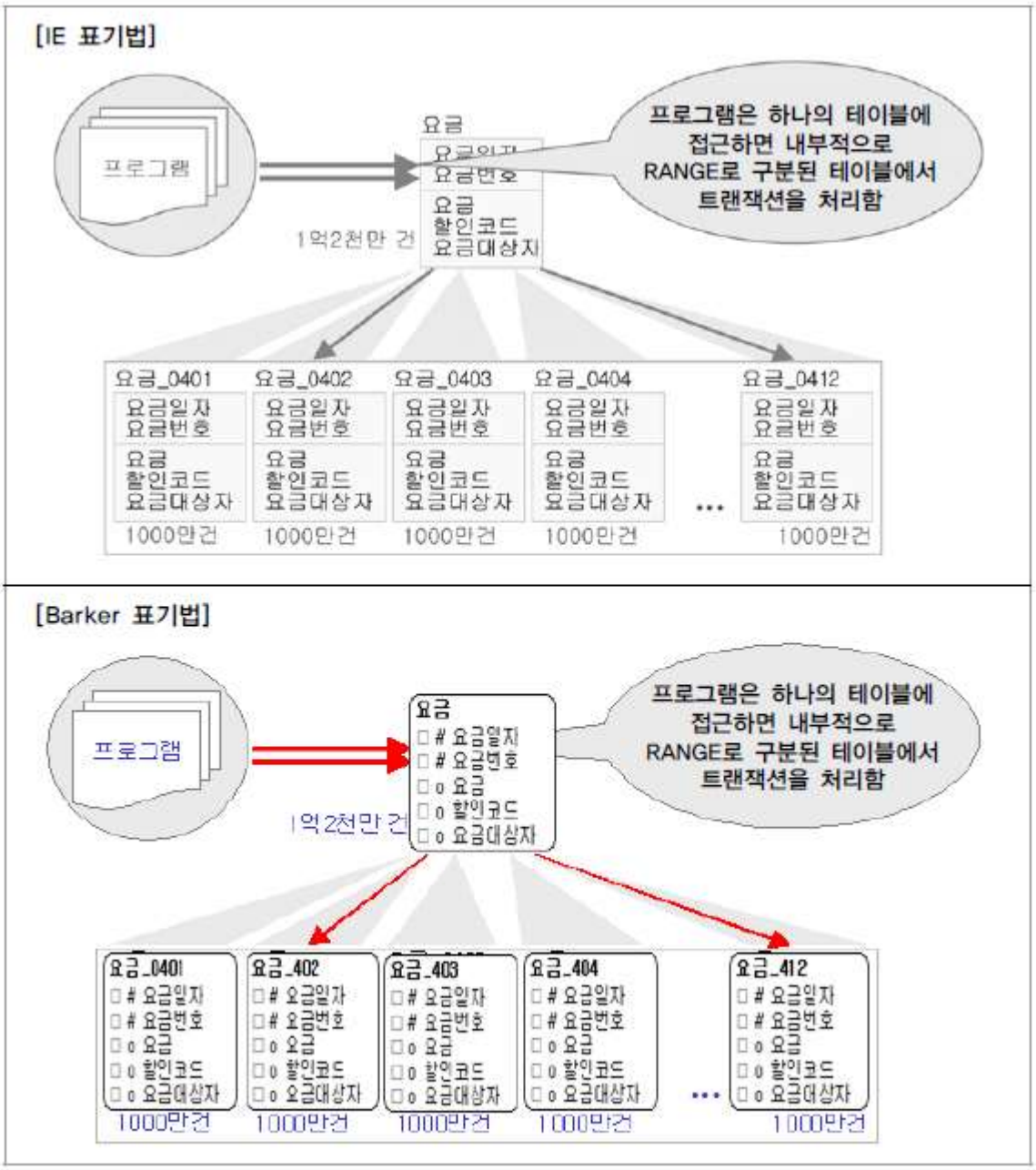
3. 대량 데이터 저장 및 처리로 인해 성능

테이블에 많은 양의 데이터가 예상될 경우 파티셔닝을 적용하거나 PK 에 의해 테이블을 분할하는 방법을 적용할 수 있다. Oracle 의 경우 크게 LIST PARTITION(특정값 지정), RANGE PARTITION(범위), HASH PARTITION(해쉬적용), COMPOSITE PARTITION(범위와 해쉬가 복합) 등이 가능하다.

데이터량이 몇 천만건을 넘어서면 아무리 서버사양이 훌륭하고 인덱스를 잘 생성해준다고 하더라고 SQL 문장의 성능이 나오지 않는다. 이 때는 논리적으로는 하나의 테이블로 보이지만 물리적으로 여러 개의 테이블스페이스에 쪼개어 저장될 수 있는 구조의 파티셔닝을 적용하도록 한다.

가. RANGE PARTITION 적용

다음은 요금테이블에 PK 가 요금일자+요금번호로 구성되어 있고 데이터건수가 1 억 2 천만 건인 대용량 테이블의 경우이다. 하나의 테이블로는 너무 많은 데이터가 존재하므로 인해 성능이 느린 경우에 해당된다. 이 때 요금의 특성상 항상 월단위로 데이터 처리를 하는 경우가 많으므로 PK 인 요금일자의 년+월을 이용하여 12 개의 파티션 테이블을 만들었다. 하나의 파티션 테이블당 평균 1,000 만 건의 데이터가 있다고 가정한다.



[그림 I-2-23] 파티셔닝의 적용-RANGE

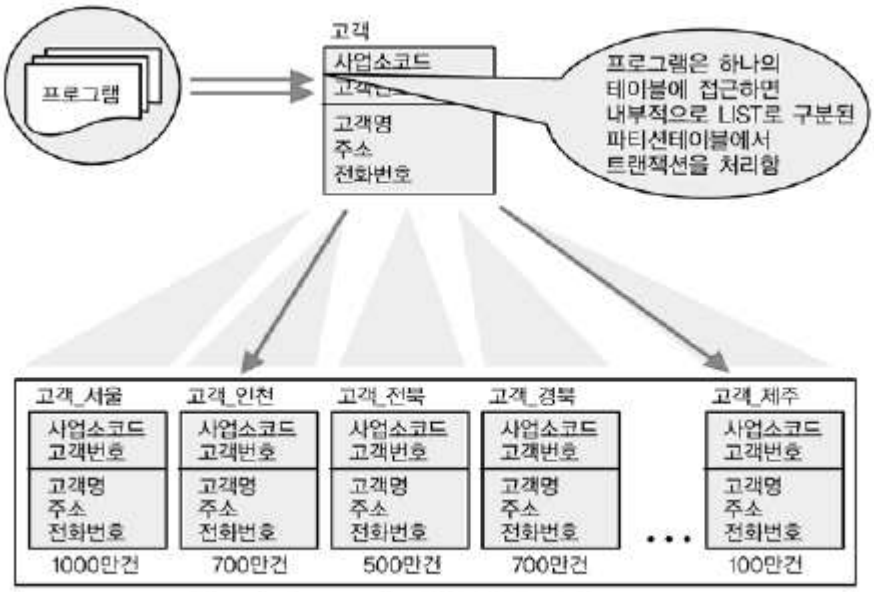
SQL 문장을 처리할 때는 마치 하나의 테이블처럼 보이는 요금 테이블을 이용하여 처리하면 되지만 DBMS 내부적으로는 SQL WHERE 절에 비교된 요금일자에 의해 각 파티션에 있는 정보를 찾아가므로 평균 1,000 만 건의 데이터가 있는 곳을 찾아도 되어 성능이 개선될 수 있다.

가장 많이 사용하는 파티셔닝의 기준이 RANGE PARTITION 이다. 대상 테이블이 날자 또는 숫자값으로 분리가 가능하고 각 영역별로 트랜잭션이 분리된다면 RANGE PARTITION 을 적용한다. 또한 RANGE PARTITION 은 데이터보관주기에 따라 테이블에 데이터를 쉽게 지우는 것이 가능하므로(파티션 테이블을 DROP 하면 되므로) 데이터보관주기에 다른 테이블관리가 용이하다.

나. LIST PARTITION 적용

지점, 사업소, 사업장, 핵심적인 코드값 등으로 PK 가 구성되어 있고 대량의 데이터가 있는 테이블이라면 값 각각에 의해 파티셔닝이 되는 LIST PARTITION 을 적용할 수 있다.

[그림 Ⅰ-2-24]의 예는 고객이라고 하는 테이블에 데이터가 1억 건이 있는데 하나의 테이블에서 데이터를 처리하기에는 SQL 문장의 성능이 저하되어 지역을 나타내는 사업소코드별로 LIST PARTITION 을 적용한 예이다.



[그림 Ⅰ-2-24] 파티셔닝의 적용-LIST

LIST PARTITION 은 대용량 데이터를 특정값에 따라 분리 저장할 수는 있으나 RANGE PARTITION 과 같이 데이터 보관주기에 따라 쉽게 삭제하는 기능은 제공될 수 없다.

다. HASH PARTITION 적용

기타 HASH PARTITION 은 지정된 HASH 조건에 따라 해쉥 알고리즘이 적용되어 테이블이 분리되며 설계자는 테이블에 데이터가 정확하게 어떻게 들어갔는지 알 수 없다. 역시 성능향상을 위해 사용하며 데이터 보관주기에 따라 쉽게 삭제하는 기능은 제공될 수 없다.

데이터량이 대용량이 되면 파티셔닝의 적용은 필수적으로 파티셔닝 기준을 나눌 수 있는 조건에 따라 적절한 파티셔닝을 방법을 선택하여 성능을 향상 시키도록 한다.

4. 테이블에 대한 수평분할/수직분할의 절차

테이블에 대한 수평분할/수직분할에 대한 결정은 다음의 4 가지 원칙을 적용하면 된다.

1) 데이터 모델링을 완성한다.
2) 데이터베이스 용량산정을 한다.
3) 대량 데이터가 처리되는 테이블에 대해서 트랜잭션 처리 패턴을 분석한다.
4) 칼럼 단위로 집중화된 처리가 발생하는지, 로우단위로 집중화된 처리가 발생되는지 분석하여 집중화된 단위로 테이블을 분리하는 것을 검토한다.

용량산정은 어느 테이블에 데이터의 양이 대용량이 되는지 분석하는 것이다. 특정 테이블의 용량이 대용량인 경우 칼럼의 수가 너무 많은 지 확인한다. 칼럼의 수가 많은 경우 트랜잭션의 특성에 따라 테이블을 1:1 형태로 분리할 수 있는지 검증하면 된다. 칼럼의 수가 적지만 데이터용량이 많아 성능저하가 예상이 되는 경우 테이블에 대해 파티셔닝 전략을 고려하도록 한다. 이 때 임의로 파티셔닝할 것인지 데이터가 발생되는 시간에 따라 파티셔닝을 할 것인지를 설명된 기준에 따라 적용하면 된다.

05.데이터베이스 구조와 성능

1. 슈퍼타입/서브타입 모델의 성능고려 방법

가. 슈퍼/서브타입 데이터 모델의 개요

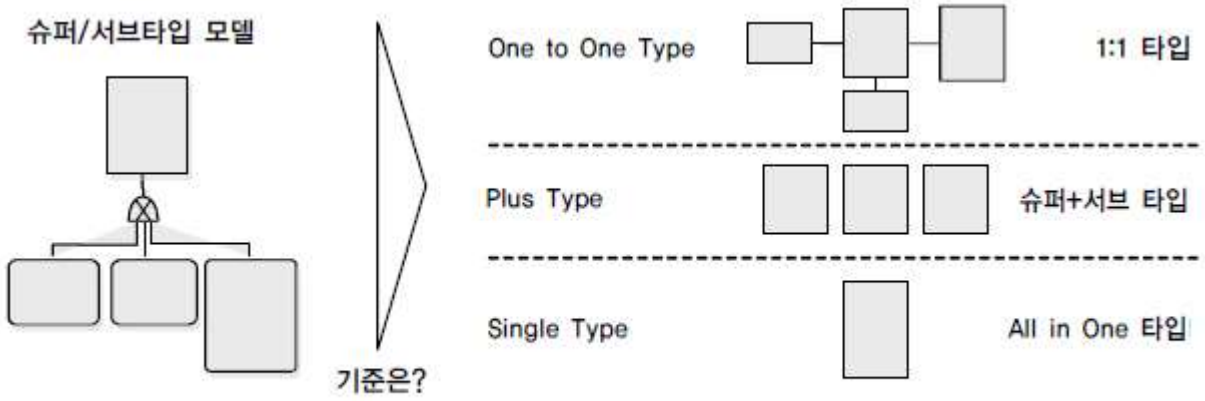
Extended ER 모델이라고 부르는 이른바 슈퍼/서브타입 데이터 모델은 최근에 데이터 모델링을 할 때 자주 쓰이는 모델링 방법이다. 이 모델이 자주 쓰이는 이유는 업무를 구성하는 데이터의 특징을 공통과 차이점의 특징을 고려하여 효과적으로 표현할 수 있기 때문이다. 즉, 공통의 부분을 슈퍼타입으로 모델링하고 공통으로부터 상속받아 다른 엔터티와 차이가 있는 속성에 대해서는 별도의 서브엔터티로 구분하여 업무의 모습을 정확하게 표현하면서 물리적인 데이터 모델로 변환을 할 때 선택의 폭을 넓힐 수 있는 장점이 있다. 이러한 장점 때문에 많은 프로젝트에서 슈퍼/서브타입을 활용한 데이터 모델의 사례가 증가하고 있다.

당연히 슈퍼/서브타입의 데이터 모델은 논리적인 데이터 모델에서 이용되는 형태이고 분석/설계단계를 구분하자면, 분석단계에서 많이 쓰이는 모델이다. 따라서 물리적인 데이터 모델을 설계하는 단계에서는 슈퍼/서브타입 데이터 모델을 일정한 기준에 의해 변환을 해야 한다. 그런데 실제로 프로젝트 현장에서는 이것을 변환하는 방법에 대해 정확한 노하우가 없기 때문에 막연하게 1:1 로 변환하거나 아니면 하나의 테이블로 구성해 버리는 현상이 나타난다.

물리적인 데이터 모델이 성능을 고려한 데이터 모델이 되어야 한다는 점을 고려하면 이렇게 막연하게 슈퍼/서브타입을 아무런 기준없이 변환하는 것 자체가 성능이 저하될 수 있는 위험이 있음을 기억해야 한다.

나. 슈퍼/서브타입 데이터 모델의 변환

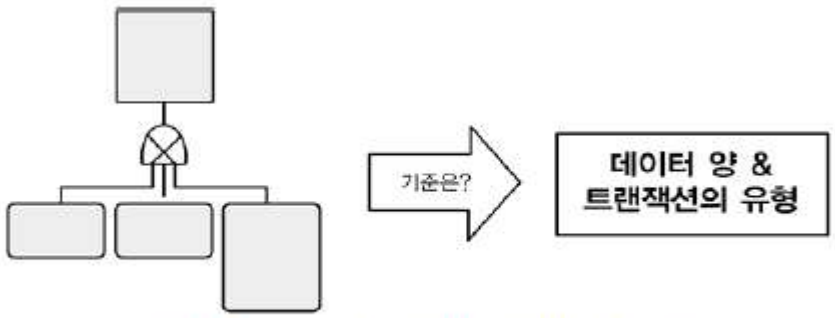
성능을 고려한 슈퍼타입과 서브타입의 모델 변환의 방법을 알아보면 [그림 Ⅰ-2-25]와 같다.



[그림 Ⅰ-2-25] 슈퍼타입과 서비스타입의 변환

슈퍼/서브타입에 대한 변환을 잘못하면 성능이 저하되는 이유는 트랜잭션 특성을 고려하지 않고 테이블이 설계되었기 때문이다. 이것을 3 가지 경우의 수로 정리하면 설명하면 다음과 같다. 1) 트랜잭션은 항상 일괄로 처리하는데 테이블은 개별로 유지되어 Union 연산에 의해 성능이 저하될 수 있다. 2) 트랜잭션은 항상 서브타입 개별로 처리하는데 테이블은 하나로 통합되어 있어 불필요하게 많은 양의 데이터가 집약되어 있어 성능이 저하되는 경우가 있다. 3) 트랜잭션은 항상 슈퍼+서브 타입을 공통으로 처리하는데 개별로 유지되어 있거나 하나의 테이블로 집약되어 있어 성능이 저하되는 경우가 있다.

해당 테이블에 발생되는 성능이 중요한 트랜잭션이 빈번하게 처리되는 기준에 따라 테이블을 설계해야 이러한 성능저하 현상을 예방할 수 있음을 기억해야 한다. 슈퍼/서브타입을 성능을 고려한 물리적인 데이터 모델로 변환하는 기준은 데이터 양과 해당 테이블에 발생되는 트랜잭션의 유형에 따라 결정된다.



[그림 Ⅰ-2-26] 슈퍼/서브타입 변환 기준

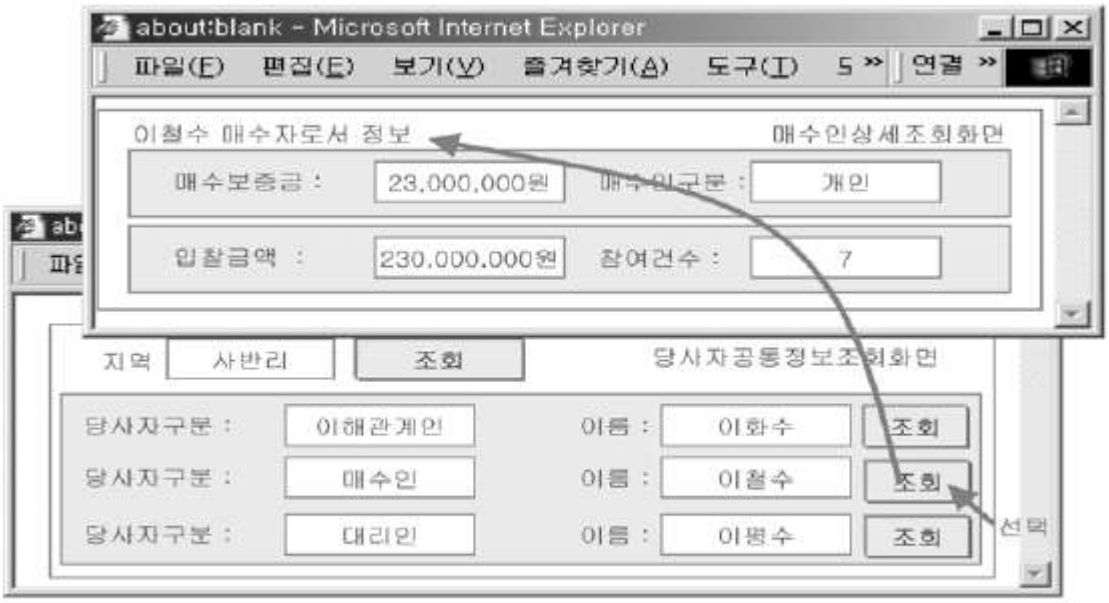
데이터의 양은 데이터량이 소량일 경우 성능에 영향을 미치지 않기 때문에 데이터처리의 유연성을 고려하여 가급적 1:1 관계를 유지하는 것이 바람직하다. 그러나 데이터용량이 많아지는 경우 그리고 해당 업무적인 특징이 성능에 민감한 경우는 트랜잭션이 해당 테이블에 어떻게 발생되는지에 따라 3 가지 변환방법을 참조하여 상황에 맞게 변환하도록 해야 한다.

다. 슈퍼/서브 타입 데이터 모델의 변환기술

논리적인 데이터 모델에서 설계한 슈퍼타입/서브타입 모델을 물리적인 데이터 모델로 전환할 때 주로 어떤 유형의 트랜잭션이 발생하는지 검증해야 한다. 물론 데이터량이 아주 작다면, 예를 들어 10 만 건도 되지 않는다면 그리고 시스템을 운영하는 중에도 증가하지 않는다면 트랜잭션의 성격을 고려하지 않고 전체를 하나의 테이블로 묶어도 좋은 방법이다.

그러나 데이터량이 많이 존재하고 지속적으로 증가하는 양도 많다면 슈퍼타입/서브타입에 대해 물리적인 데이터 모델로 변환하는 세 가지 유형에 대해 세심하게 적용을 해야 한다.

1) 개별로 발생되는 트랜잭션에 대해서는 개별 테이블로 구성
업무적으로 발생되는 트랜잭션이 슈퍼타입과 서브타입 각각에 대해 발생하는 것이다.
[그림 Ⅰ-2-27]의 업무화면을 보면 공통으로 처리하는 슈퍼타입테이블인 당사자 정보를 미리 조회하고 원하는 내용을 클릭하면 거기에 따라서 서브타입인 세부적인 정보 즉 이해관계인, 매수인, 대리인에 대한 내용을 조회하는 형식이다. 즉 슈퍼타입이 각 서브타입에 대해 기준역할을 하는 형식으로 사용할 때 이러한 유형의 트랜잭션이 발생이 된다.



[그림 Ⅰ-2-27] 트랜잭션 유형

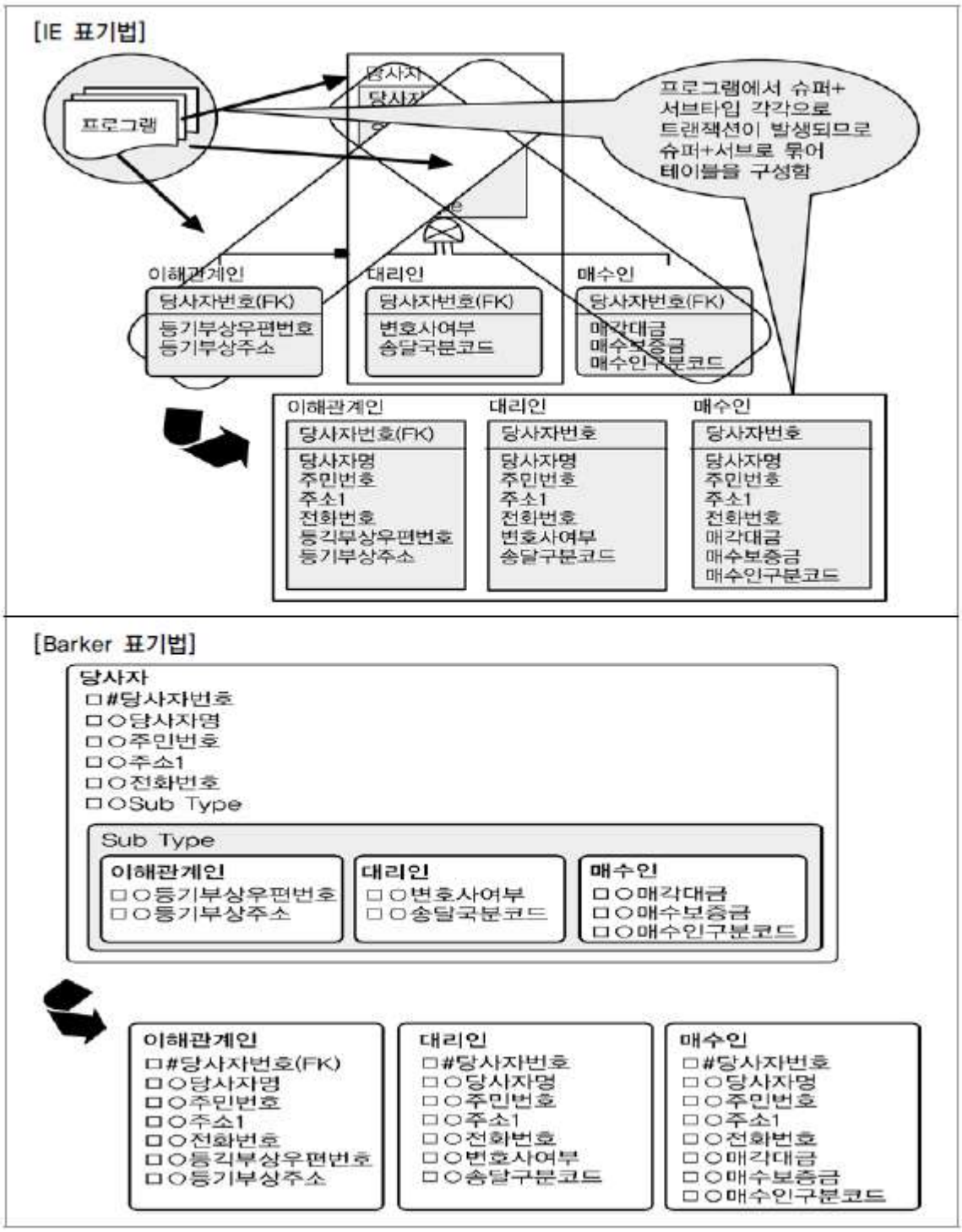
위와 같이 슈퍼타입과 서브타입각각에 대해 독립적으로 트랜잭션이 발생이 되면 슈퍼타입에도 꼭 필요한 속성만을 가지게 하고 서브타입에도 꼭 필요한 속성 및 자신이 타입에 맞는 데이터만 가지게 하기 위해서 모두 분리하여 1:1 관계를 갖도록 한다.

실전프로젝트에서는 데이터량이 대용량으로 존재하는 경우에 공통으로 이용하는 슈퍼타입의 속성의 수가 너무 많아져 디스크 I/O 가 많아지는 것을 방지하기 위해 위와 같이 각각을 1:1 관계로 가져가는 경우도 있다.

2) 슈퍼타입+서브타입에 대해 발생되는 트랜잭션에 대해서는 슈퍼타입+서브타입 테이블로 구성
만약 대리인이 10 만 건, 매수인 500 만 건, 이해관계인 500 만 건의 데이터가 존재한다고 가정하고 슈퍼타입과 서브타입이 모두 하나의 테이블로 통합되어 있다고 가정하자. 매수인, 이해관계인에 대한 정보는 배제하고 10 만 건뿐인 대리인에 대한 데이터만 처리할 경우 다른 테이블과 같이 데이터가 1 천 10 만 건이 저장되어 있는 곳에서 처리해야 하므로 불필요한 성능저하 현상이 유발된다. 즉 대리인에 대한 처리가 개별적으로 많이 발생하는데 매수인과 이해관계인의 데이터까지 포함되어 있으므로 최대 10 만 건을 읽어 처리할 수 있는 업무가 최대 1 천 10 만 건을 읽어 처리하는 경우가 발생될 수 있다.

이와 같이 슈퍼타입과 서브타입을 묶어 트랜잭션이 발생하는 업무특징을 가지고 있을 때에는 다음 데이터 모델과 같이 슈퍼타입+각서브타입을 하나로 묶어 별도의 테이블로 구성하는 것이 효율적이다.

업무적인 특성상 실전 프로젝트에서 슈퍼타입/서브타입모델은 위와 같이 각각이 슈퍼타입+서브타입으로 묶여 구성하는 경우가 많다.



[그림 Ⅰ-2-28] Plus Type 변환

3) 전체를 하나로 묶어 트랜잭션이 발생할 때는 하나의 테이블로 구성

대리인 10 만 건, 매수인 500 만 건, 이해관계인 500 만 건의 데이터가 존재한다고 하더라고 데이터를 처리할 때 대리인, 매수인, 이해관계인을 항상 통합하여 처리한다고 하면 테이블을 개별로 분리해야 불필요한 조인을 유발하거나 불필요한 UNION ALL 과 같은 SQL 구문이 작성되어 성능이 저하된다. 비록 슈퍼타입과 서브타입의 테이블들을 하나로 묶었을 때 각각의 속성별로 제약사항(NULL/NOT NULL, 기본값, 체크값)을 정확하게 지정하지 못할지라도 대용량이고 성능향상이 필요하다면 하나의 테이블로 묶어서 만들어 준다.

3 가지 전개 방식이 아주 간단한 원리 같은데 이것도 실전 프로젝트에서 적용하면 쉽지 않은 경우가 많이 나타난다. 때로는 각각의 유형이 혼합되어 있는 경우도 있다. 혼합된 트랜잭션 유형이 있는 경우는 많이 발생하는 트랜잭션 유형에 따라 구성하면 된다.

라. 슈퍼/서브타입 데이터 모델의 변환타입 비교

[표 Ⅰ-2-4] 슈퍼/서브타입 데이터 모델 변환타입 비교

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 선택 방법	개별 테이블로 접근이 많은 경우 선택	슈퍼+서브 형식으로 데이터를 처리하는 경우 선택	전체를 일괄적으로 처리하는 경우 선택

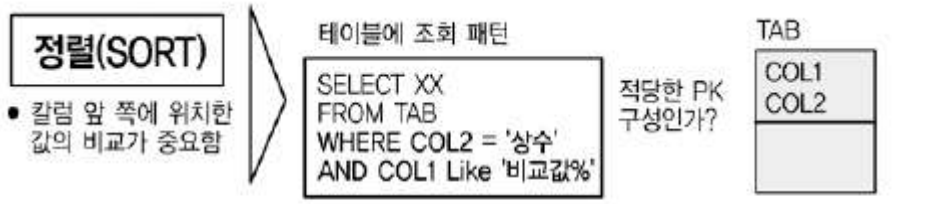
[표 Ⅰ-2-4]와 같이 각 성능이 좋을 수도 나쁠 수도 있기 때문에 변환모델의 선택은 철저하게 데이터베이스에 발생되는 트랜잭션의 유형에 따라 선택을 해야 한다.

2. 인덱스 특성을 고려한 PK/FK 데이터베이스 성능향상

가. PK/FK 칼럼 순서와 성능개요

가. PK/FK 칼럼 순서와 성능개요 데이터를 조회할 때 가장 효과적으로 처리될 수 있도록 접근경로를 제공하는 오브젝트가 바로 인덱스이다. 일반적으로 데이터베이스 테이블에서는 균형 잡힌 트리 구조의 B*Tree 구조를 많이 사용한다. 우리는 B*Tree 구조의 내부 알고리즘까지는 알 필요가 없더라도 그 구조를 이용할 때 정렬되어 있는 특징으로 인해 데이터베이스 설계에 이 특징에 따라 설계에 반영해야 할 요소에 대해서는 반드시 알고 있어야 좋은 데이터 모델을 만들어 낼 수 있게 된다.

프로젝트에서 PK/FK 설계는 업무적 의미로도 매우 중요한 의미를 가지고 있지만 데이터를 접근할 때 경로를 제공하는 성능의 측면에서도 중요한 의미를 가지고 있기 때문에 성능을 고려한 데이터베이스 설계가 될 수 있도록 설계단계 말에 칼럼의 순서를 조정할 필요가 있다.
일반적으로 프로젝트에서는 PK/FK 칼럼 순서의 중요성을 인지하지 못한 채로 데이터 모델링이 되어 있는 그 상태대로 바로 DDL 을 생성함으로써 데이터베이스 데이터처리 성능에 문제를 유발하는 경우가 빈번하게 발생이 된다.



테이블에 발생되는 트랜잭션의 조회 패턴에 따라 PK/FK 칼럼의 순서를 조정해야 함

[그림 Ⅰ-2-29] 인덱스 특성에 따른 PK/FK순서

간단한 것 같지만 실전 프로젝트에서는 아주 중요한 내용이 바로 PK 순서이다. 성능저하 현상이 많은 부분이 PK 가 여러 개의 속성으로 구성된 복합식별자 일 때 PK 순서에 대해 별로 고려하지 않고 데이터 모델링을 한 경우에 해당된다.<>br 특히 물리적인 데이터 모델링 단계에서는 스스로 생성된 PK 순서 이외에 다른 엔터티로부터 상속받아 발생되는 PK 순서까지 항상 주의하여 표시하도록 해야 한다. PK 는 해당 해당테이블의 데이터를 접근할 가장 빈번하게 사용되는 유일한 인덱스(Unique Index)를 모두 자동 생성한다. PK 순서를 결정하는 기준은 인덱스 정렬구조를 이해한 상태에서 인덱스를 효율적으로 이용할 수 있도록 PK 순서를 지정해야 한다. 즉 인덱스의 특징은 여러 개의 속성이 하나의 인덱스로 구성되어 있을 때 앞쪽에 위치한 속성의 값이 비교자로 있어야 인덱스가 좋은 효율을 나타낼 수 있다. 앞쪽에 위치한 속성 값이 가급적 '=' 아니면 최소한 범위 'BETWEEN' '<>' 가 들어와야 인덱스를 이용할 수 있는 것이다.

데이터 모델링 때 결정한 PK 순서와는 다르게 DDL 문장을 통해 PK 순서를 다르게 생성할 수도 있다. 그러나 대부분의 프로젝트에서는 데이터 모델의 PK 순서에 따라 그대로 PK 를 생성한다. 만약 다르게 생성한다고 하더라도 데이터 모델과 데이터베이스 테이블의 구조가 다른 것처럼 보여 유지보수에 어려움이 많을 것이다. 또한 FK 라고 하더라도 데이터를 조회할 때 조인의 경로를 제공하는 역할을 수행하므로 FK 에 대해서는 반드시 인덱스를 생성하도록 하고 인덱스 칼럼의 순서도 조회의 조건을 고려하여 접근이 가장 효율적인 칼럼 순서대로 인덱스를 생성하도록 주의해야 한다.

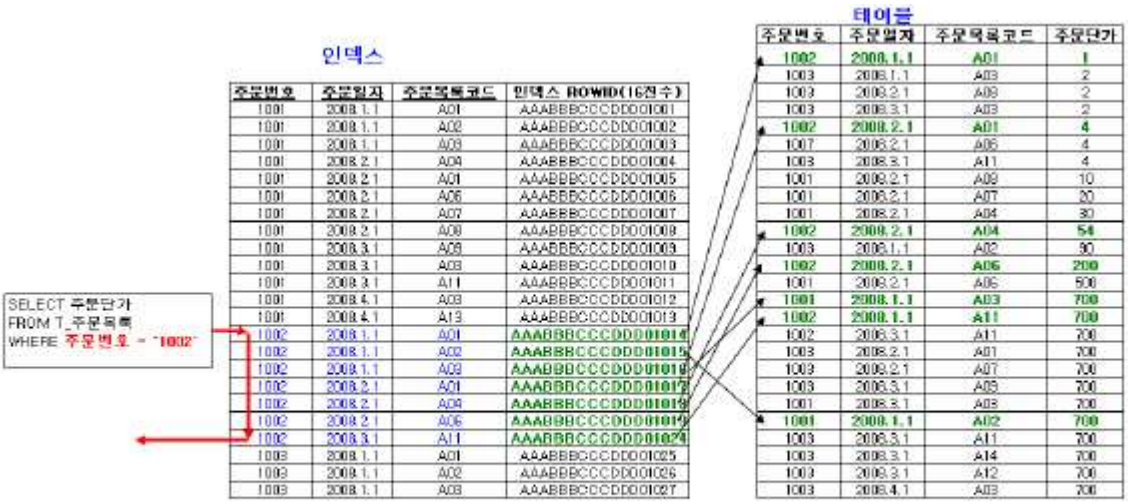
나. PK 칼럼의 순서를 조정하지 않으면 성능이 저하 이유

먼저 데이터 모델링에서 엔터티를 설계하면 그에 따라 DDL 이 생성이 되고 생성된 DDL 에 따라 인덱스가 생성된다. 이 때 우리가 알아야 할 구조는 인덱스의 정렬구조에 해당된다.
(그림 Ⅰ-2-30)은 인덱스의 정렬구조가 생성되는 구조를 보여주고 있다.

[그림 Ⅰ-2-30] 데이터 모델과 인덱스 구조

(그림 Ⅰ-2-30)에서 보면 테이블에서 데이터 모델의 PK 순서에 따라 DDL 이 그대로 생성이 되고 테이블의 데이터가 주문번호가 가장먼저 정렬되고 그에 따라 주문일자가 정렬이 되고 마지막으로 주문목록코드가 정렬되는 것을 알 수 있다. 이러한 정렬 구조로 인해 데이터를 접근하는 트랜잭션의 조건에 따라 다른 인덱스 접근방식을 보여주게 된다.

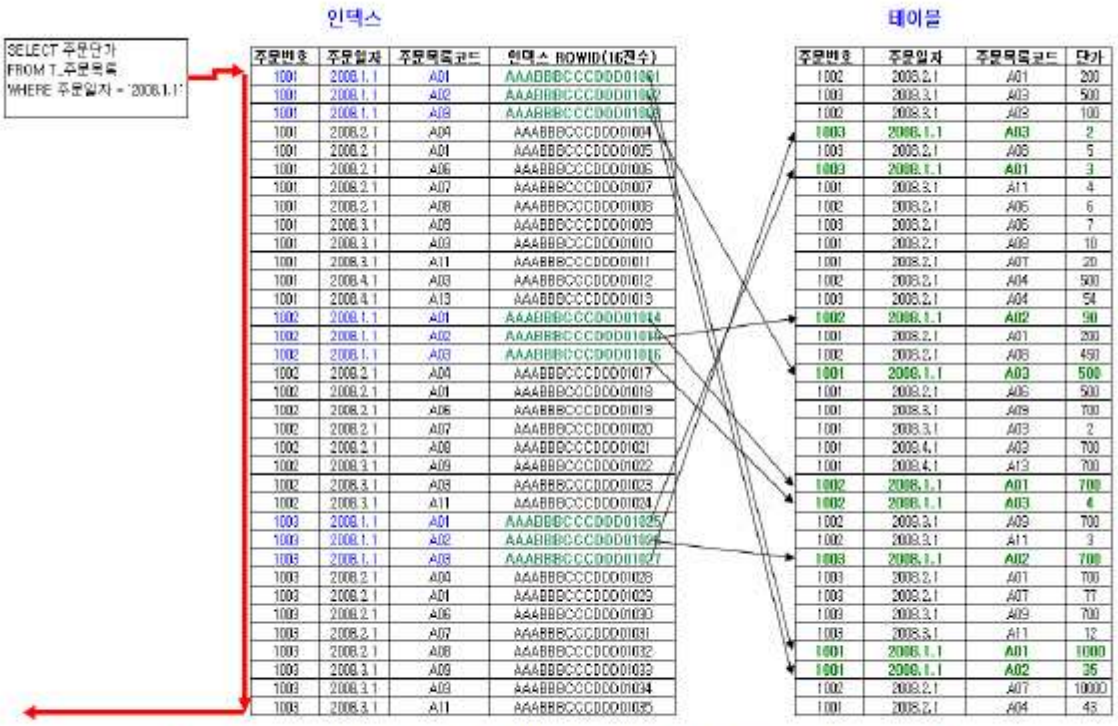
위와 같은 인덱스의 정렬 구조에서 SQL 구문의 조건에 따라 인덱스를 처리하는 범위가 달라지게 된다. 맨 앞에 있는 인덱스 칼럼에 대해 조회 조건이 들어올 때 데이터를 접근하는 방법은 [그림 Ⅰ-2-31]과 같다.



맨 앞에 있는 인덱스 칼럼에 조회 조건이 들어올 때 Scan방법

[그림 Ⅰ-2-31] 조건에 따른 인덱스 Scan구조(1)

인덱스의 정렬된 첫 번째 칼럼에 비교가 되었기 때문에 순차적으로 데이터를 찾아가게 된다. 맨 앞에 있는 칼럼이 제외된 상태에서 데이터를 조회 할 경우 데이터를 비교하는 범위가 매우 넓어지게 되어 성능 저하를 유발하게 된다.



맨 앞에 있는 인덱스 칼럼에 조회 조건이 들어오지 않을 때 Scan방법

[그림 Ⅰ-2-32] 조건에 따른 인덱스 Scan구조(2)

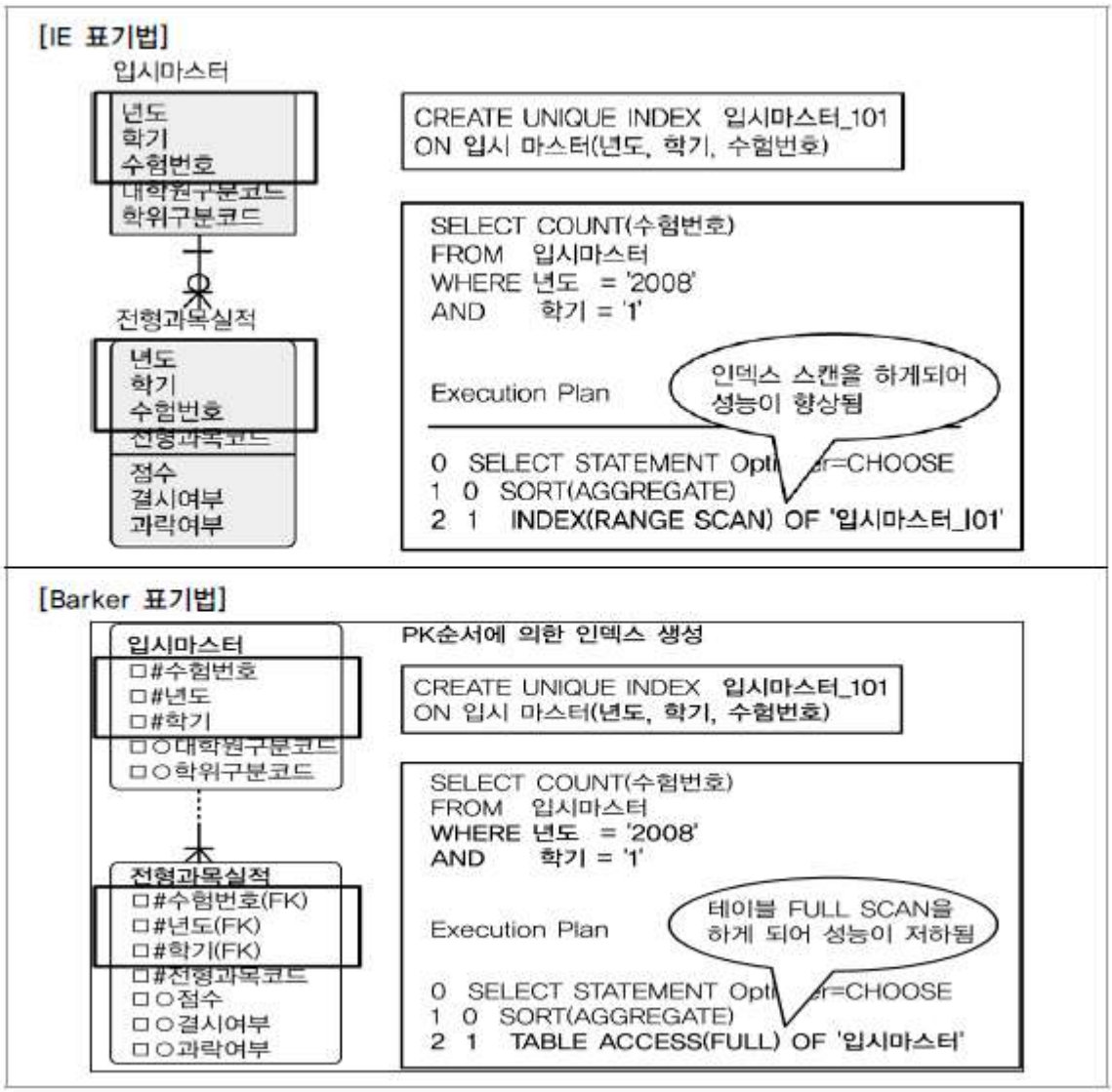
[그림 Ⅰ-2-32]의 예에서는 주문번호에 대한 비교값이 들어오지 않으므로 인해 인덱스 전체를 읽어야만 원하는 데이터를 찾을 수 있게 된다. 이러한 이유로 인덱스를 읽고 테이블 블록에서 읽어 처리하는데 I/O가 많이 발생하게 되므로 옵티마이저는 차라리 테이블에 가서 전체를 읽는 방식으로 처리하게 된다.

이러한 모습으로 인덱스의 정렬구조를 이해한 상태에서 인덱스에 접근하는 접근유형을 비교해보면 어떠한 인덱스를 태워야 하는지 어떠한 조건이 들어와야 데이터를 처리하는 양을 줄여 성능을 향상시킬 수 있는지 알 수 있게 된다.

정리하면, PK 의 순서를 인덱스 특징에 맞게 고려하지 않고 바로 그대로 생성하게 되면, 테이블에 접근하는 트랜잭션의 특징에 효율적이지 않은 인덱스가 생성되어 있으므로 인덱스의 범위를 넓게 이용하거나 Full Scan 을 유발하게 되어 성능이 저하된다고 정리할 수 있다.
다음은 실전 프로젝트에서 발생되는 예를 통해 PK 나 FK 의 성능저하 사항을 알아보도록 한다.

다. PK 순서를 잘못 지정하여 성능이 저하된 경우 - 간단한 오류

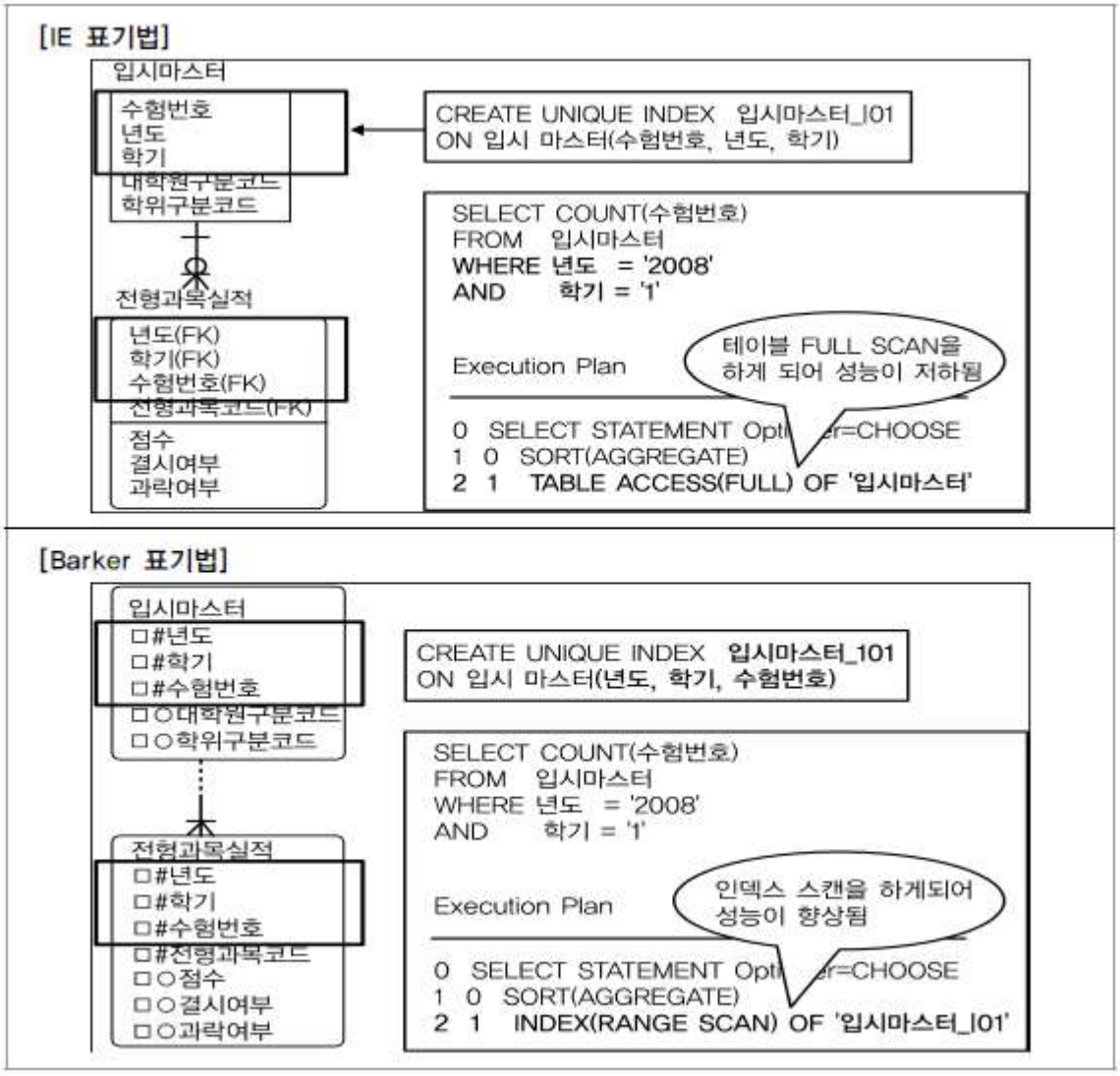
입시마스터라는 테이블의 PK 는 수험번호+년도+학기로 구성되어 있고 전형과목실적 테이블은 입시마스터 테이블에서 상속받은 수험번호+년도+학기에 전형과목코드로 PK 가 구성되어 있는 복합식별자 구조의 테이블이다. 입시마스터에는 200 만 건의 데이터가 있고 학사는 4 학기로 구성되어 있고 데이터는 5 년간 보관되어 있다. 그러므로 한 학기당 평균 2 만 건의 데이터가 있다고 가정하자.



[그림 Ⅰ-2-33] PK순서에 의한 인덱스 생성 개선 전

이 테이블 구조에서 다음과 같은 SQL 구문이 실행되면 입시마스터 테이블에 있는 인덱스 입시마스터_I01 을 이용할 수 있을까?

```
SELECT COUNT(수험번호) FROM 입시마스터 WHERE 년도 = '2008' AND 학기 = '1'
```



[그림 Ⅰ-2-34] PK순서에 의한 인덱스 생성 개선 후

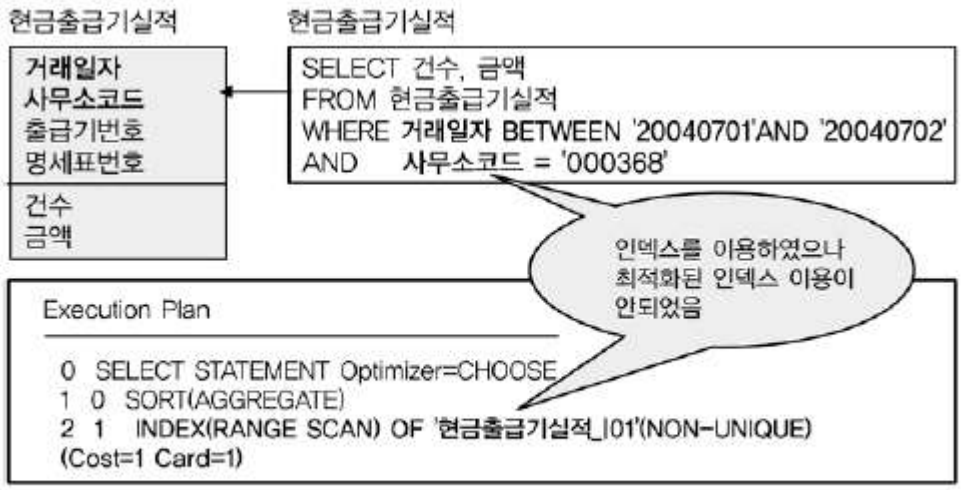
입시마스터_I01 인덱스가 수험번호+년도+학기 중 수험번호에 대한 값이 WHERE 절에 들어오지 않으므로 FULL TABLE SCAN 이 발생, 200 만 건의 데이터를 모두 읽게 되어 성능이 저하되었다.
입시마스터 테이블에 데이터를 조회할 때 년도와 학기에 대한 내용이 빈번하게 들어오므로 [그림 Ⅰ-2-34]와 같이 PK 순서를 변경함으로써 인덱스를 이용 가능하도록 할 수 있다. 즉, 생성된 인덱스가 정상적으로 이용이 되어 평균 2 만 건의 데이터를 처리함으로써 성능이 개선된 모습이다.

라. PK 순서를 잘못 지정하여 성능이 저하된 경우 - 복잡한 오류

현금출급기실적의 PK 는 거래일자+사무소코드+출급기번호+명세표번호로 되어 있는데 대부분의 SQL 문장에서는 조회를 할 때 사무소코드가 '=' 로 들어오고 거래일자에 대해서는 'BETWEEN' 조회를 하고 있다. 이 때 SQL 은 정상적으로 인덱스를 이용할 수 있지만 인덱스 효율이 떨어져 성능이 저하되는 경우에 해당된다.

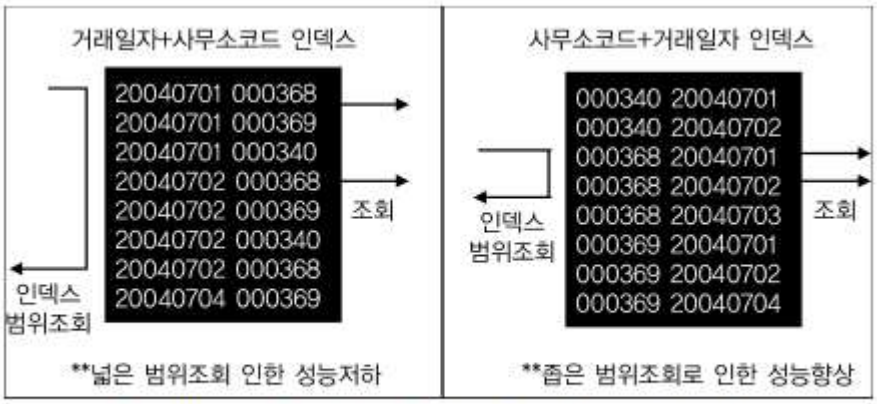
해당 테이블에 발생하는 SQL 은 다음과 같이 작성되었다.

```
SELECT 건수, 금액 FROM 현금출급기실적 WHERE 거래일자 BETWEEN '20040701' AND '20040702' AND 사무소코드 = '000368'
```



[그림 Ⅰ-2-35] PK순서에 의한 인덱스 생성

실행계획을 분석해 보면 인덱스가 정상적으로 이용되었기 때문에 SQL 문장은 튜닝이 잘된 것으로 착각할 수 있다. 문제는 인덱스를 이용하기는 하는데 얼마나 효율적으로 이용하는지 검증이 필요하다. 아래 그림은 거래일자+사무소코드 순서로 인덱스를 구성한 경우와 사무소코드+거래일자 순서로 인덱스를 구성한 경우 데이터를 처리하는 범위의 차이를 보여주는 그림이다. 거래일자+사무소코드로 구성된 그림을 보면 BETWEEN 비교를 한 거래일자 '20040701' 이 인덱스의 앞에 위치하기 때문에 범위가 넓어졌고 사무소코드+거래일자로 구성된 인덱스의 경우 '=' 비교를 한 사무소코드 '000368' 이 인덱스 앞에 위치하여 범위가 좁아졌다.



[그림 Ⅰ-2-36] PK순서에 의한 인덱스 이용 범위

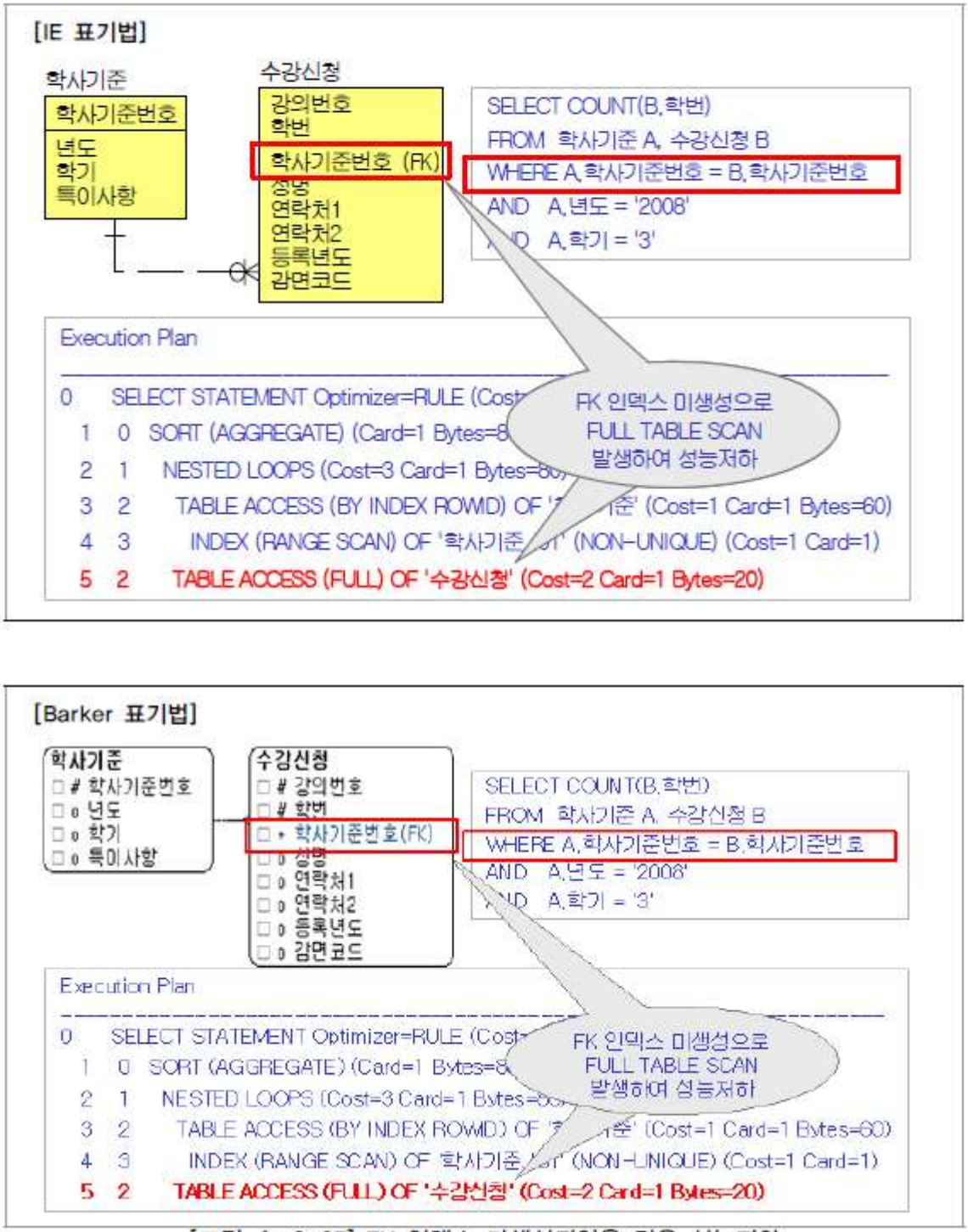
그러므로 이 경우 인덱스순서를 고려하여 데이터 모델의 PK 순서를 거래일자+사무소코드+출급기번호+명세표번호에서 사무소코드+거래일자+출급기번호+명세표번호로 수정하여 성능을 개선할 수 있다. 물론 테이블의 PK 구조를 그대로 둔 상태에서 인덱스만 하나 더 만들어도 성능은 개선될 수 있다. 이 때 이미 만들어진 PK 인덱스가 전혀 사용되지 않는다면 입력, 수정, 삭제시 불필요한 인덱스로 인해 더 성능이 저하되어 좋지 않다. 최적화된 인덱스 생성을 위해 PK 순서변경을 통한 인덱스 생성이 바람직하다.

그러면 테이블의 PK 의 속성이 A, B 가 있을 때 A+B 형태로도 빈번하게 조회가 되고 B+A 로도 빈번하게 조회되는 경우에는 어떻게 할 것인가? 이 때는 좀 더 자주 이용되는 조회의 형태대로 PK 순서를 구성하여 이용하게 하고 순서를 바꾼 인덱스를 추가로 생성하는 것이 필요하다.

3. 물리적인 테이블에 FK 제약이 걸려있지 않을 경우 인덱스 미생성으로 성능저하

물리적인 테이블에 FK 를 사용하지 않아도 데이터 모델 관계에 의해 상속받은 FK 속성들은 SQL WHERE 절에서 조인으로 이용되는 경우가 많이 있으므로 FK 인덱스를 생성해야 성능이 좋은 경우가 빈번하다.

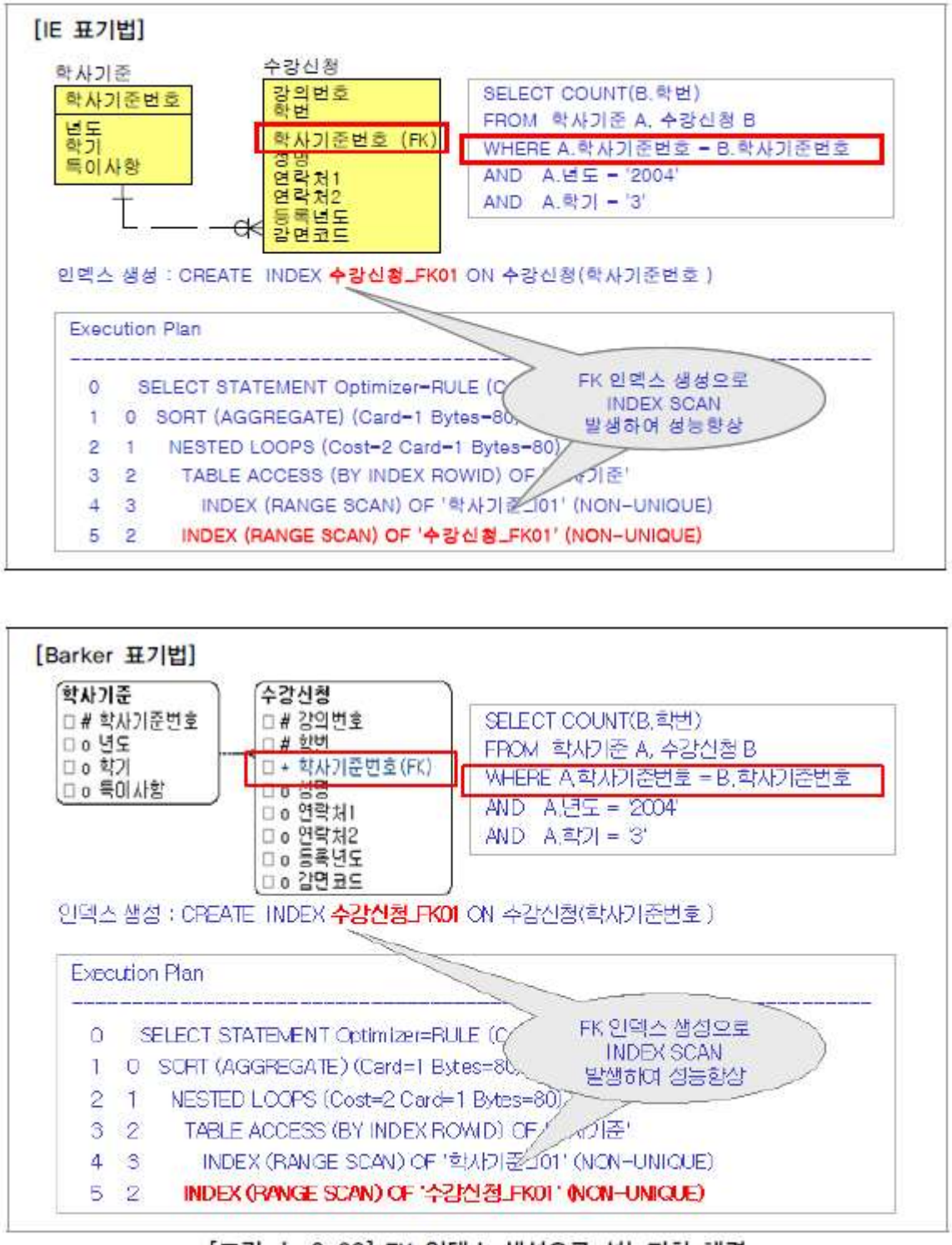
다음 그림은 학사기준과 수강신청에 대한 데이터 모델이다. 물리적인 테이블에는 두 테이블사이에 FK 참조무결성 관계가 걸려 있지 않는다고 가정한다. 또한 학사기준에는 데이터가 5 만 건이 있고 수강신청에 데이터가 500 만 건이 있다고 가정하자.



[그림 I-2-37] FK 인덱스 미생성되었을 경우 성능저하

비록 수강신청 테이블에 있는 학사기준번호가 SQL WHERE 절에 비교자로 들어오지는 않았지만 수강신청 테이블에서 상속받은 학사기준번호에 대해 인덱스를 생성하지 않으므로 인해 학사기준과 수

강신청 테이블이 조인이 되면서 500 만 건의 수강신청 테이블이 FULL TABLE SCAN 이 발생되어 성능이 저하되었다. 이 때는 수강신청 테이블에 FK 인덱스를 생성하여 성능을 개선할 수 있다.



[그림 I-2-38] FK 인덱스 생성으로 성능저하 해결

비록 물리적으로 학사기준과 수강신청이 연결되어 있지 않다고 하더라도 학사기준으로부터 상속받은 FK 에 대해 FK 인덱스를 생성함으로써 SQL 문장이 조인이 발생할 때 성능저하를 예방할 수 있다.

FK 인덱스를 적절하게 설계하여 구축하지 않았을 경우 개발초기에는 데이터량이 얼마 되지 않아 성능저하가 나타나지 않다가 시스템을 오픈하고 데이터량이 누적될수록 SQL 성능이 나빠짐으로 인해 데이터베이스서버에 심각한 장애현상을 초래하는 경우가 많이 있다.

그러므로 물리적인 테이블에 FK 제약 걸었을 때는 반드시 FK 인덱스를 생성하도록 하고 FK 제약이 걸리지 않았을 경우에는 FK 인덱스를 생성하는 것을 기본정책으로 하되 발생되는 트랜잭션에 의해 거의 활용되지 않았을 때에만 FK 인덱스를 지우는 방법으로 하는 것이 적절한 방법이 된다.

06.분산 데이터베이스와 성능

1. 분산 데이터베이스의 개요

1990 년대에는 데이터베이스를 분산하여 저장하고 그것을 하나의 데이터베이스로 인식하여 사용하는 기술은 아주 난이도가 높은 고급기술로 인식되었다. 2000 년도에 클라우드 컴퓨팅, SOA 를 인식하듯 분산 데이터베이스를 인식하고 연구 · 도입하려는 기업이 많았었다. DBMS 의 기능이 강해지고 네트워크 속도가 빨라지면서 분산 데이터베이스가 초기에 예상한 만큼 확산되지는 않았지만, 여전히 많은 데이터베이스는 네트워크를 통한 데이터베이스 간의 공유체계를 통해 분산 데이터베이스를 활용하고 있다.

분산데이터베이스의 정의는 다음과 같다.

- 여러 곳으로 분산되어있는 데이터베이스를 하나의 가상 시스템으로 사용할 수 있도록 한 데이터베이스
- 논리적으로 동일한 시스템에 속하지만, 컴퓨터 네트워크를 통해 물리적으로 분산되어 있는 데이터들의 모임. 물리적 Site 분산, 논리적으로 사용자 통합 · 공유

즉, 분산 데이터베이스는 데이터베이스를 연결하는 빠른 네트워크 환경을 이용하여 데이터베이스를 여러 지역 여러 노드로 위치시켜 사용성/성능 등을 극대화 시킨 데이터베이스라고 정의할 수 있다.

2. 분산 데이터베이스의 투명성(Transparency)

분산데이터베이스가 되기 위해서는 6 가지 투명성(Transparency)을 만족해야 한다.

1) 분할 투명성 (단편화) : 하나의 논리적 Relation 이 여러 단편으로 분할되어 각 단편의 사본이 여러 site 에 저장
2) 위치 투명성 : 사용하려는 데이터의 저장 장소 명시 불필요. 위치정보가 System Catalog 에 유지되어야 함
3) 지역사상 투명성 : 지역 DBMS 와 물리적 DB 사이의 Mapping 보장. 각 지역시스템 이름과 무관한 이름 사용 가능
4) 중복 투명성 : DB 객체가 여러 site 에 중복 되어 있는지 알 필요가 없는 성질
5) 장애 투명성 : 구성요소(DBMS, Computer)의 장애에 무관한 Transaction 의 원자성 유지
6) 병행 투명성 : 다수 Transaction 동시 수행시 결과의 일관성 유지, Time Stamp, 분산 2 단계 Locking 을 이용 구현

전통적인 분산 데이터베이스 구축과 같이, 분산 환경의 데이터베이스를 위와 같은 특징 모두를 만족하면서 구축하는 사례는 최근에는 드물다. 최근에는 분산 환경의 데이터베이스를 구축하기보다 통합하여 데이터베이스를 구축하는 사례가 더 많이 있다. 그럼에도 불구하고 위의 분산 환경의 데이터베이스를 업무적인 특징 및 지역적인 특징에 따라 적절하게 활용하기만 하면, 다양한 장점을 제공하는 특징을 가지고 있기 때문에 대량 데이터처리의 지역적 처리나 글로벌 처리 등에서는 분산 데이터베이스가 유용하게 활용되고 있다.

3. 분산 데이터베이스의 적용 방법 및 장단점

가. 분산 데이터베이스 적용방법

분산 환경의 데이터베이스를 성능이 우수하게 현장에서 가치 있게 사용하는 방법은 업무의 흐름을 보고 업무구성에 따른 아키텍처 특징에 따라 데이터베이스를 구성하는 것이다. 단순히 분산 환경에서 데이터베이스를 구축하는 것이 목적이 아니라, 업무의 특징에 따라 데이터베이스 분산구조를 선

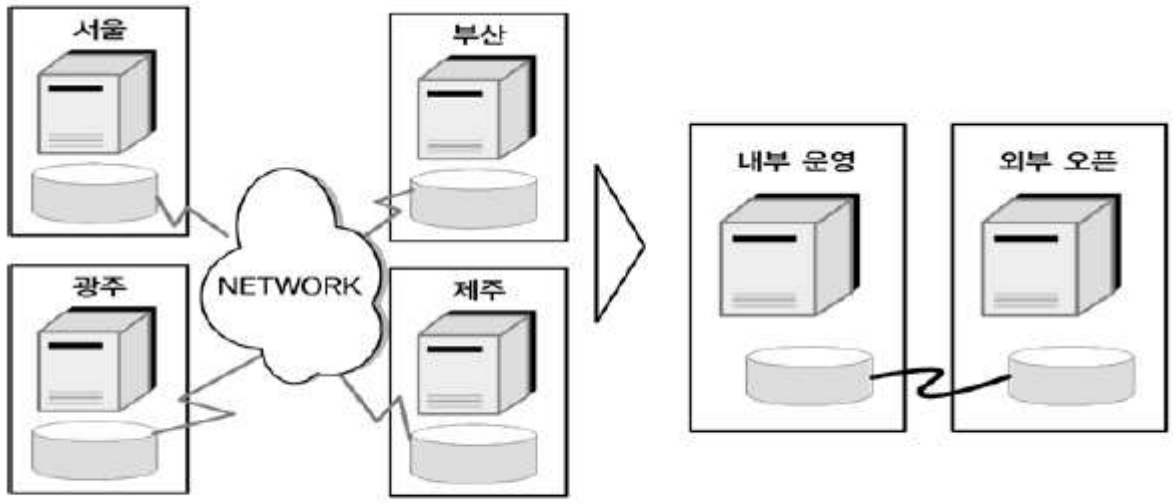
택적으로 설계하는 능력이 필요한 것이다. 이러한 측면만을 보았을 때는 데이터베이스 분산설계라는 측면보다는 데이터베이스 구조설계(아키텍처)라는 의미로 이해해도 무방할 것이다.

나. 분산 데이터베이스 장단점

장점	단점
- 지역 자치성, 점증적 시스템 용량 확장 - 신뢰성과 가용성 - 효용성과 융통성 - 빠른 응답 속도와 통신비용 절감 - 데이터의 가용성과 신뢰성 증가 - 시스템 규모의 적절한 조절 - 각 지역 사용자의 요구 수용 증대	- 소프트웨어 개발 비용 - 오류의 잠재성 증대 - 처리 비용의 증대 - 설계, 관리의 복잡성과 비용 - 불규칙한 응답 속도 - 통제의 어려움 - 데이터 무결성에 대한 위협

4. 분산 데이터베이스의 활용 방향성

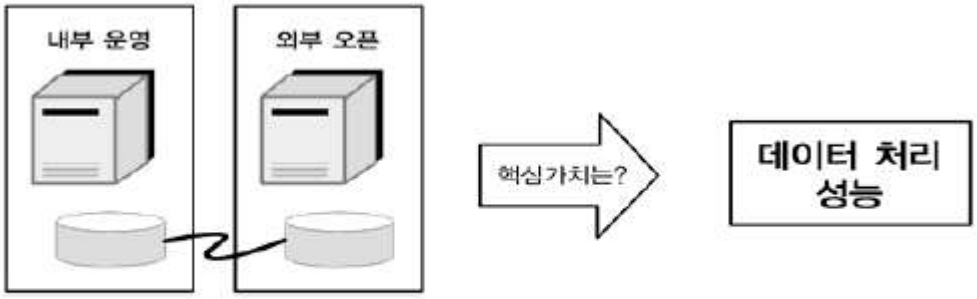
분산 데이터베이스는 업무적인 기능이 다양해지고 데이터의 양이 기하급수적으로 증가하는 최근 데이터베이스 환경에서 적용하는 고급화된 기술이다. 업무적인 특징에 따라 분산 데이터베이스를 활용하는 기술이 필요하다.



[그림 Ⅰ-2-39] 분산설계 방향성

5. 데이터베이스 분산구성의 가치

데이터를 분산 환경으로 구성하였을 때 가장 핵심적인 가치는 바로 통합된 데이터베이스에서 제공할 수 없는 빠른 성능을 제공한다는 점이다. 원거리 또는 다른 서버에 접속하여 처리하므로 인해 발생되는 네트워크 부하 및 트랜잭션 집중에 따른 성능 저하의 원인을 분산된 데이터베이스 환경을 구축하므로 빠른 성능을 제공하는 것이 가능해 진다. 바로 이 점 때문에 분산 환경의 데이터베이스를 구축하게 되는 것이다.



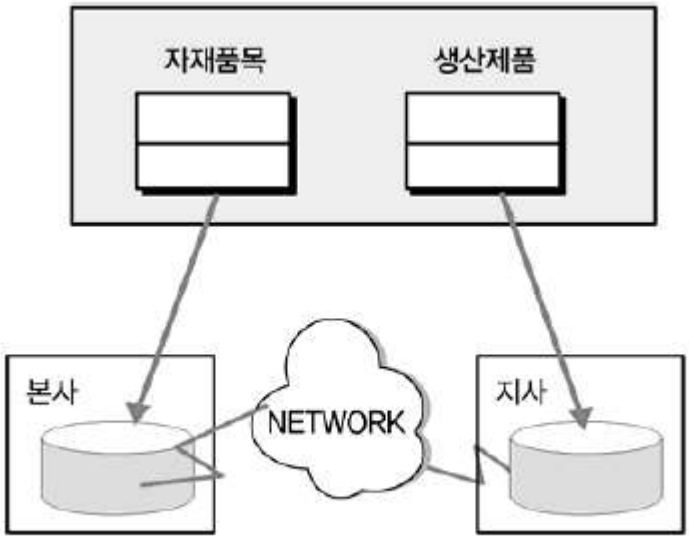
[그림 Ⅰ-2-40] 분산 데이터베이스 핵심가치

6. 분산 데이터베이스의 적용 기법

데이터베이스의 분산의 종류에는 테이블 위치 분산과 테이블 분할 분산, 테이블 복제 분산, 테이블 요약 분산 전략이 있다. 그 중에서도 가장 많이 사용하는 방식의 테이블의 복제 분할 분산의 방법이고 이 방법은 성능이 저하되는 많은 데이터베이스에서 가장 유용하게 적용할 수 있는 기술적인 방법이 된다. 분산 환경으로 데이터베이스를 설계하는 방법은 일단 통합 데이터 모델링을 하고 각 테이블별로 업무적인 특징에 따라 지역 또는 서버별로 테이블을 분산 배치나 복제 배치하는 형태로 설계할 수 있다.

가. 테이블 위치 분산

테이블 위치 분산은 테이블의 구조는 변하지 않는다. 또한 테이블이 다른 데이터베이스에 중복되어 생성되지도 않는다. 다만 설계된 테이블의 위치를 각각 다르게 위치시키는 것이다. 예를 들어, 자재품목은 본사에서 구입하여 관리하고 각 지사별로 자재품목을 이용하여 제품을 생산한다고 하면 [그림 Ⅰ-2-41]과 같이 데이터베이스를 본사와 지사단위로 분산시킬 수 있다.



[그림 Ⅰ-2-41] 테이블별 위치 분산

[그림 Ⅰ-2-41]의 분산방법은 설계된 테이블 각각이 지역별로 분산되어 생성되는 경우이다.
각각의 테이블마다 위치가 다르게 지정되어야 한다면 [그림 Ⅰ-2-42]의 표와 같이 각각 테이블마다 위치를 표기하여 테이블을 생성하도록 한다.

위치 \ 테이블	자재품목	생산제품	협력회사	사원	부서
본사	•		•		•
지사		•		•	

[그림 Ⅰ-2-42] 테이블별 위치 분산

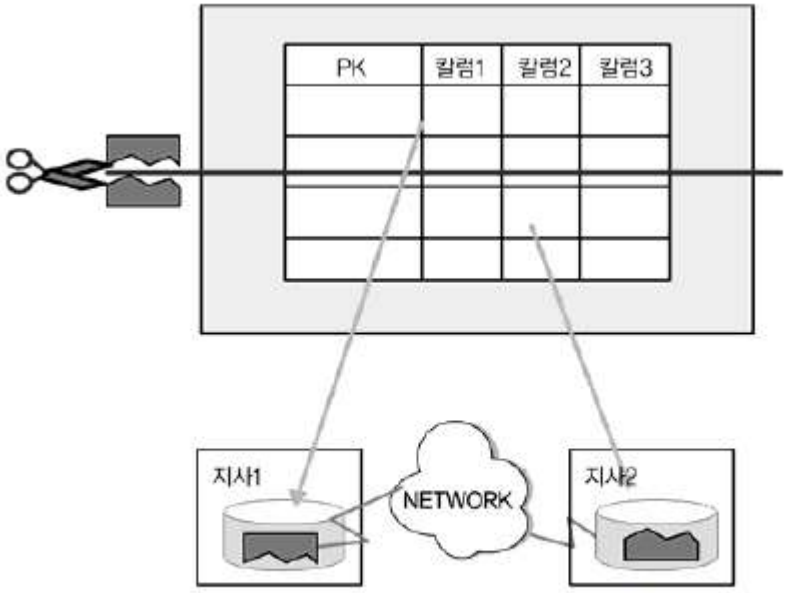
테이블별 위치 분산은 정보를 이용하는 형태가 각 위치별로 차이가 있을 경우에 이용한다. 테이블의 위치가 위치별로 다르므로 테이블의 위치를 파악할 수 있는 도식화된 위치별 데이터베이스 문서가 필요하다.

나. 테이블 분할(Fragmentation) 분산

테이블 분할 분산은 단순히 위치만 다른 곳에 두는 것이 아니라 각각의 테이블을 쪼개어 분산하는 방법이다. 테이블을 분할하여 분산하는 방법은 테이블을 나누는 기준에 따라 두 가지로 구분된다. 첫 번째는 테이블의 로우(Row)단위로 분리하는 수평분할(Horizontal Fragmentation)이 있고 두 번째는 테이블을 칼럼(Column) 단위로 분할하는 수직분할(Vertical Fragmentation)이 있다.

- 수평분할(Horizontal Fragmentation)

지사(Node)에 따라 테이블을 특정 칼럼의 값을 기준으로 로우(Row)를 분리한다. 칼럼은 분리되지 않는다. 모든 데이터가 각 지사별로 분리되어 있는 형태를 가지고 있다. 각 지사에 있는 데이터와 다른 지사에 있는 데이터와 항상 배타적으로 존재하며 한군데 집합시켜도 Primary key 에 의해 중복이 발생하지않는다.



[그림 Ⅰ-2-43] 테이블 분할 분산 – 수평분할

이와 같이 수평분할을 이용하는 경우는 각 지사(Node)별로 사용하는 로우(Row)가 다를 때 이용한다. 데이터를 수정할 때는 타 지사에 있는 데이터를 원칙적으로 수정하지 않고 자신의 데이터에 대해서 수정하도록 한다. 각 지사에 존재하는 테이블에 대해서 통합처리를 해야 하는 경우는 조인(JOIN)이 발생하여 성능 저하가 예상되므로 통합처리 프로세스가 많은지를 먼저 검토한 이후에 많지 않은 경우에 수평분할을 해야 한다.

데이터가 지사별로 별도로 존재하므로 중복은 발생하지 않는다. 대신 타 지사에 있는 데이터가 지사구분이 변경되면 단순히 수정이 발생하는 것 이외에 변경된 지사로 데이터를 이송해야 한다. 한 시점에는 한 지사(Node)에서 하나의 데이터만이 존재하므로 데이터의 무결성은 보장되는 형태이다. 지사(Node)별로 데이터베이스를 운영하는 경우는 데이터베이스가 속한 서버가 지사(Node)에 존재하

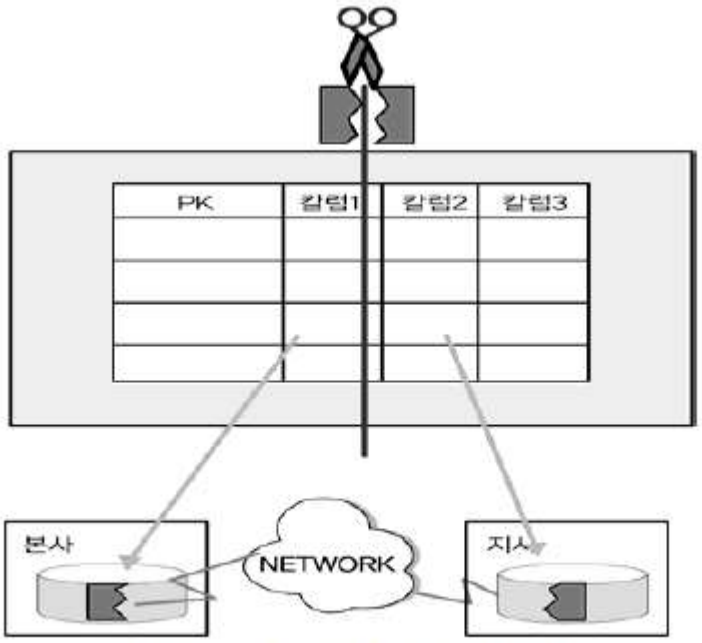
던지 아니면 본사에 통합해서 존재하건 간에 데이터베이스 테이블들은 수평 분할하여 존재한다. 지사별로 운영하는 테이블들의 예를 들어보자. 테이블 고객, 생산제품, 협력회사, 사원, 부서 테이블이 지사 1 과 지사 2 에서 일의 시작과 끝이 항상 다르게 발생한다고 하면 각 테이블은 지사별로 수평 분할하여 [그림 Ⅰ-2-44]의 표와 같이 생성되게 된다.

테이블 / 위치	고객	생산제품	협력회사	사원	부서
지사 1	◓	◓	◓	◓	◓
지사 2	◒	◒	◒	◒	◒

[그림 Ⅰ-2-44] 테이블 수평분할

- 수직분할(Vertical Fragmentation)

지사(Node)에 따라 테이블 칼럼을 기준으로 칼럼(Row)을 분리한다. 로우(Row) 단위로는 분리되지 않는다. 모든 데이터가 각 지사별로 분리되어 있는 형태를 가지고 있다. 칼럼을 기준으로 분할하였기 때문에 각각의 테이블에는 동일한 Primary Key 구조와 값을 가지고 있어야 한다. 지사별로 쪼개어진 테이블들을 조합하면 Primary Key 가 동일한 데이터의 조합이 가능해야 하며 하나의 완전한 테이블이 구성되어야 한다. 데이터를 한군데 집합시켜 놓아도 동일한 Primary Key 는 하나로 표현하면 되므로 데이터 중복은 발생되지 않는다.



[그림 Ⅰ-2-45] 테이블 수직분할

예를 들어 제품의 재고량은 각 지사별로 관리하고 제품에 대한 단가는 본사에서 관리한다고 하면 본사 테이블에는 제품번호, 단가가 존재하고 지사에는 제품번호, 재고량이 존재한다. 이를 본사와 지사단위로 분리된 칼럼의 모습을 도식화 하면 다음과 같이 나타난다.

테이블 / 위치	제품	분할 칼럼
본사	◐	제품번호, 단가
지사	◑	제품번호, 재고량

[그림 Ⅰ-2-46] 테이블 수직분할 모델링

테이블의 전체 칼럼 데이터를 보기 위해서는 각 지사(Node)별로 흩어져 있는 테이블들을 조인(JOIN)하여 가져와야 하므로 가능하면 통합하여 처리하는 프로세스가 많은 경우에는 이용하지 않도록 한

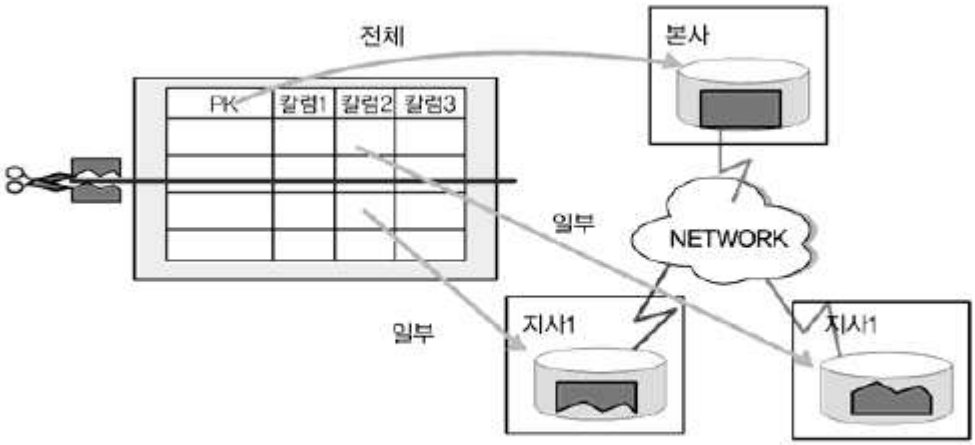
다. 일반적으로 실제 프로젝트에서는 이와 같이 칼럼을 쪼개는 테이블의 수직분할 분산 환경을 구성하는 사례는 드물다.

다. 테이블 복제(Replication) 분산

테이블 복제(Replication) 분산은 동일한 테이블을 다른 지역이나 서버에서 동시에 생성하여 관리하는 유형이다.
마스터 데이터베이스에서 테이블의 일부의 내용만 다른 지역이나 서버에 위치시키는 부분복제(Segment Replication)가 있고 마스터 데이터베이스의 테이블의 내용을 각 지역이나 서버에 존재시키는 광역복제(Broadcast Replication)가 있다.

- 부분복제(Segment Replication)

통합된 테이블을 한군데(본사)에 가지고 있으면서 각 지사별로는 지사에 해당된 로우(Row)를 가지고 있는 형태이다. 지사에 존재하는 데이터는 반드시 본사에 존재하게 된다. 즉 본사의 데이터는 지사데이터의 합이 되는 것이다. 각 지사에서 데이터 처리가 용이할 뿐만 아니라 전체 데이터에 대한 통합처리도 본사에 있는 통합 테이블을 이용하게 되므로 여러 테이블에 조인(JOIN)이 발생하지 않는 빠른 작업 수행이 가능해진다.



[그림 Ⅰ-2-47] 테이블 복제 분산 - 부분복제

[그림 Ⅰ-2-47]을 보면, 본사 데이터베이스에 있는 테이블에는 테이블의 전체 내용이 들어가고 각 지사 데이터베이스에 있는 테이블에는 지사별로 관계된 데이터만 들어가게 된다. 수평분할 분산과 마찬가지로 지사간에는 데이터의 중복이 발생하지 않으나 본사와 지사간에는 데이터의 중복이 항상 발생하게 되는 경우이다.
보통 전국에 있는 고객을 관리할 때 본사에는 전국고객에 대한 정보를 관리하고 지사에는 각 지사와 거래하는 고객정보를 관리한다. 본사의 데이터를 이용하여 통계, 이동 등을 관리하며 지사에 있는 데이터를 이용하여 지사별로 빠른 업무수행을 한다. 보통 지사에 데이터가 먼저 발생하고 본사에 데이터는 지사에 데이터를 이용하여 통합하여 발생된다.

테이블 / 위치	고객
본사	●
지사 1	◐
지사 2	◑

본사에는 전국의 고객정보를 관리하고 지사 1의 고객테이블에서는 지사 1에 속한 고객정보를 지사 2의 고객테이블에서는 지사 2에 속한 고객정보를 관리한다.

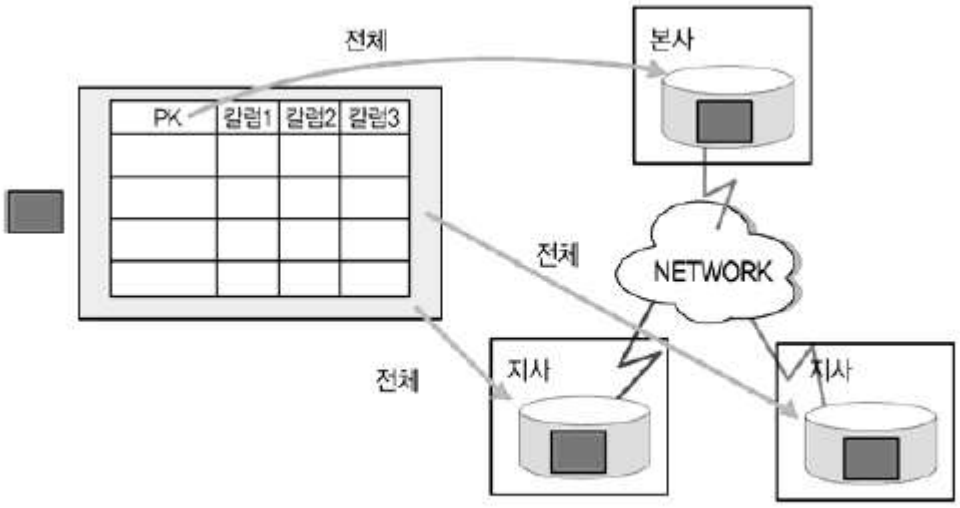
[그림 Ⅰ-2-48] 테이블 복제 분산 - 부분복제

실제 프로젝트에서 많이 사용하는 데이터베이스 분산기법에 해당한다. 각 지사별로 업무수행이 용이하고 본사에 있는 데이터를 이용하여 보고서를 출력하거나 통계를 산정하는 등 다양한 업무형태로 이용 가능하다.

다른 지역간의 데이터를 복제(Replication)하는데 많은 시간이 소요되고 데이터베이스와 서버에 부하(Load)가 발생하므로 보통 실시간(On-Line) 처리에 의해 복사하는 것보다는 야간에 배치 작업에 의해 수행되는 경우가 많이 있다.
또한 본사와 지사 양쪽 모두 데이터를 수정하여 전송하는 경우 데이터의 정합성을 일치시키는 것이 어렵기 때문에 가능하면 한쪽(지사)에서 데이터의 수정이 발생하여 본사로 복제(Replication)를 하도록 한다.

- 광역복제(Broadcast Replication)

통합된 테이블을 한군데(본사)에 가지고 있으면서 각 지사에도 본사와 동일한 데이터를 모두 가지고 있는 형태이다. 지사에 존재하는 데이터는 반드시 본사에 존재하게 된다. 모든 지사에 있는 데이터량과 본사에 있는 데이터량이 다 동일하다. 본사와 지사모두 동일한 정보를 가지고 있으므로 본사나 지사나 데이터처리에 특별한 제약을 받지는 않는다.



[그림 Ⅰ-2-49] 테이블 복제 분산 - 광역복제

예를 들어, 본사에서 코드테이블에 데이터에 대해 입력, 수정, 삭제가 발생하고 각 지사에서는 코드 데이터를 이용하는 프로세스가 발생한다. 즉 본사에서는 데이터를 관리하고 지사에서는 이 데이터를 읽어 업무프로세스를 발생시키는 것이다.

위치 \ 테이블	코드
본사	●
지사 1	●
지사 2	●

본사, 지사1, 지사2 모두 동일한 양의 코드 테이블의 데이터를 가지고 있다.

[그림 Ⅰ-2-50] 테이블 복제 분산 - 광역복제

광역복제(Broadcast Replication) 역시 실제 프로젝트에서 많이 사용하는 데이터베이스 분산기법에 해당한다. 부분복제의 경우는 지사에서 데이터에 대한 입력, 수정, 삭제가 발생하여 본사에서 이용하는 방식이 많은 반면 광역복제(Broadcast Replication)의 경우에는 본사에서 데이터가 입력, 수정, 삭제가 되어 지사에서 이용하는 형태가 차이점이다.

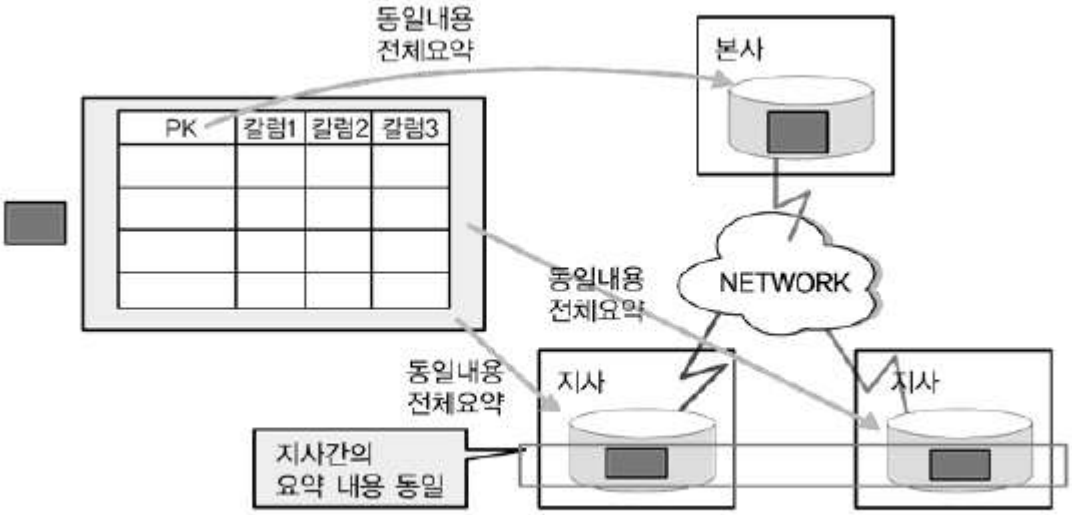
부분복제와 마찬가지로 데이터를 복제(Replication)하는데 많은 시간이 소요되고 데이터베이스와 서버에 부하(Load)가 발생하므로 보통 실시간(On-Line) 처리에 의해 복사하는 것보다는 배치에 의해 복제가 되도록 한다.

라. 테이블 요약(Summarization) 분산

테이블 요약(Summarization) 분산은 지역간에 또는 서버 간에 데이터가 비슷하지만 서로 다른 유형으로 존재하는 경우 있다. 요약의 방식에 따라, 동일한 테이블 구조를 가지고 있으면서 분산되어 있는 동일한 내용의 데이터를 이용하여 통합된 데이터를 산출하는 방식의 분석요약(Rollup Summarization)과 분산되어 있는 다른 내용의 데이터를 이용하여 통합된 데이터를 산출하는 방식의 통합요약(Consolidation Summarization)이 있다.

- 분석요약(Rollup Replication)

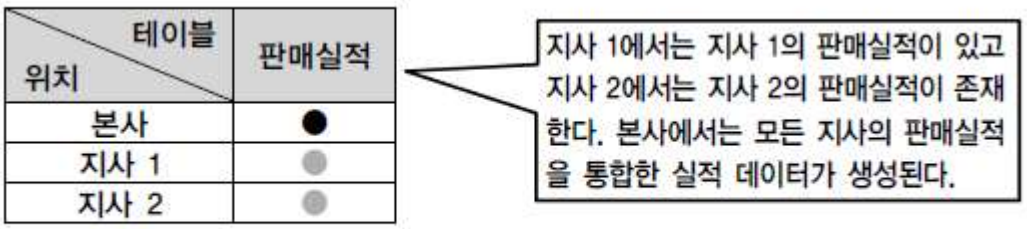
분석요약(Rollup Replication)은 각 지사별로 존재하는 요약정보를 본사에 통합하여 다시 전체에 대해서 요약정보를 산출하는 분산방법이다.



[그림 Ⅰ-2-51] 테이블 요약 분산 - 분석요약

[그림 Ⅰ-2-51]에서 보면, 테이블에 있는 모든 칼럼(Column)과 로우(Row)가 지사에도 동일하게 존재하지만, 각 지사에는 동일한 내용에 대해 지사별로 요약되어 있는 정보를 가지고 있고 본사에는 각 지사의 요약정보를 통합하여 재산출하여 전체에 대한 요약정보를 가지고 있는 것으로 표시되어 있다.

예를 들어, 제품별 판매실적이라는 테이블이 존재한다고 가정하자. 각 지사에서는 취급제품이 동일하다. 지사별로 판매된 제품에 대해서 지사별로 판매실적이 관리된다. 지사 1 과 지사 2 에도 동일한 제품이 취급이 되므로 이를 본사에서 판매실적을 집계할 경우에는 통합된 판매실적을 관리할 수 있는 것이다.

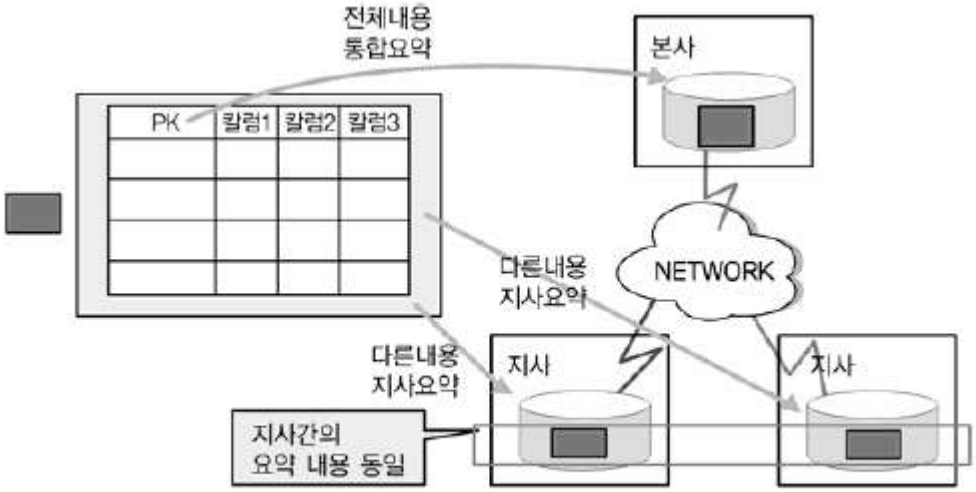


[그림 Ⅰ-2-52] 테이블 요약 분산 - 분석요약

각종 통계데이터를 산정할 경우에, 모든 지사의 데이터를 이용하여 처리하면 성능이 지연되고 각 지사 서버에 부하를 주기 때문에 업무에 장애가 발생할 수 있다. 통합 통계데이터에 대한 정보제공에 용이한 분산방법이다. 본사에 분석 요약된 테이블을 생성하고 데이터는 역시 일반 업무가 종료되는 야간에 수행하여 생성한다.

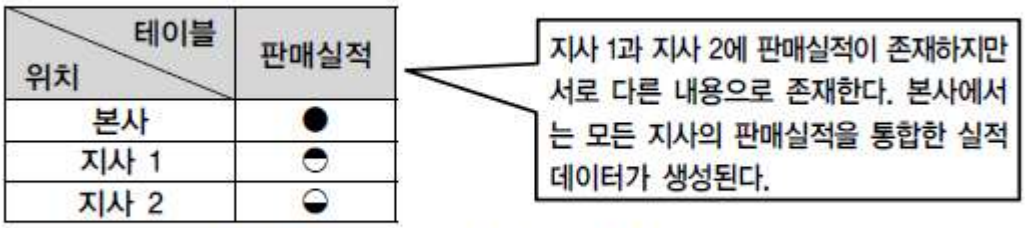
- 통합요약(Consolidation Replication)

통합요약(Consolidation Replication)은 각 지사별로 존재하는 다른 내용의 정보를 본사에 통합하여 다시 전체에 대해서 요약정보를 산출하는 분산방법이다.



[그림 Ⅰ-2-53] 테이블 요약 분산 - 통합요약

[그림 Ⅰ-2-53]에서 보면, 테이블에 있는 모든 칼럼(Column)과 로우(Row)가 지사에도 동일하게 존재하지만 각 지사에는 타지사와 다른 요약정보를 가지고 있고 본사에는 각 지사의 요약정보를 데이터를 같은 위치에 두는 것으로 통합하여 전체에 대한 요약정보를 가지고 있는 것으로 표시된다.



[그림 Ⅰ-2-54] 테이블 요약분산 - 통합요약

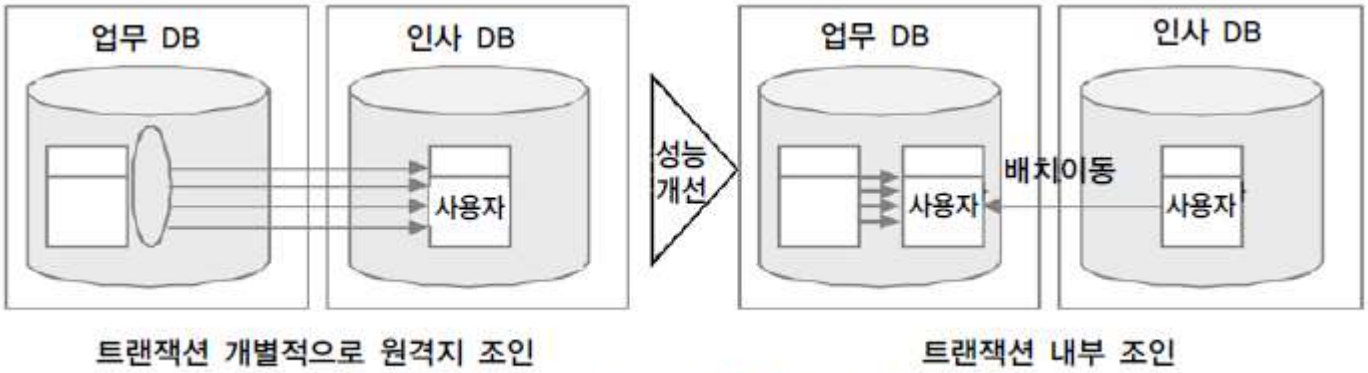
본사에 통계데이터를 산정하는 유형은 분석요약과 비슷하나 통합요약은 단지 지사에서 산출한 요약정보를 한군데 취합하여 보여주는 형태이다. 분석요약은 지사에 있는 데이터를 이용하여 본사에서 통합하여 요약 데이터를 산정하였지만 통합요약에서는 지사에서 요약한 정보를 본사에서 취합하여 각 지사별로 데이터를 비교하기 위해 이용되는 것이다.

각종 통계데이터를 산정할 경우에, 모든 지사의 데이터를 조인하여 처리하면 성능이 지연되고 각 지사 서버에 부하(LOAD)를 주기 때문에 업무에 장애가 발생할 수 있다. [그림 Ⅰ-2-54]와 같은 방법은 통합 통계데이터에 대한 정보제공에 용이한 분산방법이다. 본사에 통합 요약된 테이블을 생성하고 데이터는 역시 일반 업무가 종료되는 야간에 수행하여 생성하는 것이 일반적인 적용방법이다.

7. 분산 데이터베이스를 적용하여 성능이 향상된 사례

프로젝트를 수행할 때 단순한 분산 환경의 원리를 이해하지 않고 데이터베이스를 설계하여 성능이 저하되는 경우가 빈번하다. 특히 복제분산의 원리를 간단하게 응용하면 많은 업무적인 특성이 있는 곳에서 그 성능을 향상시켜 설계할 수 있다.

[그림 Ⅰ-2-55]는 개인정보를 관리하는 데이터베이스가 인사 데이터베이스일 때 분산이 안된 경우의 각 서버에 독립적으로 테이블이 있을 때 트랜잭션과 복제분산을 통해 테이블의 정보가 양쪽에 있을 때 트랜잭션 처리의 특성을 보여주는 그림이다. 단순한 개념도 이지만 위의 원리가 복잡한 업무처리에서 효과적으로 성능을 향상시킬 수 있음을 주목해야 한다.



[그림 Ⅰ-2-55] 업무 특성에 따른 분산환경 구성

데이터베이스 분산 설계는 다음과 같은 경우에 적용하면 효과적이다.

- 성능이 중요한 사이트에 적용해야 한다.
- 공통코드, 기준정보, 마스터 데이터 등에 대해 분산환경을 구성하면 성능이 좋아진다.
- 실시간 동기화가 요구되지 않을 때 좋다. 거의 실시간(Near Real Time)의 업무적인 특징을 가지고 있을 때도 분산 환경을 구성할 수 있다
- 특정 서버에 부하가 집중이 될 때 부하를 분산할 때도 좋다.
- 백업 사이트(Disaster Recovery Site)를 구성할 때 간단하게 분산기능을 적용하여 구성할 수 있다.