[AI초급] SQLD 자격증 코스 - 챕터 5

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- 데이터베이스의 구조와 성능에 대해 학습합니다

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

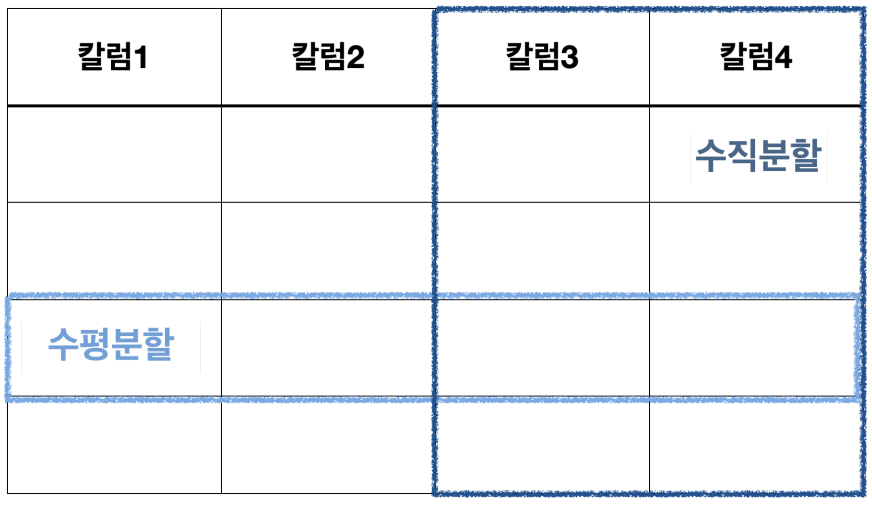
01. 대량 데이터에 따른 성능

대량 데이터에 따른 성능 문제와 향상법에 대해 학습합니다

▼ 1) 대량 데이터 발생에 따른 테이블 분할 개요

아무리 설계가 잘되어 있는 데이터 모델이라도 대량의 데이터가 하나의 테이블에 집약되어 있는 경우에는 성능 저하는 피하기 어렵습니다. 도로의 차선을 아무리 많이 만들어도 통행하는 차가 더 많아지면 어쩔 수 없는 정체가 발생하는 것과 같은 이치입니다. 일이 처리되는 양이 한 군데 몰리는 현상은 특정한 업무를 수행함에 있어 중요 업무를 온전하게 처리할 수 없게 만들기 때문에 반드시 해결해야 합니다.

테이블 단위의 분할을 통해 문제 해결이 가능한데, 수직 / 수평 분할을 통해 어떤 방식으로 문제를 해결할 수 있는지 살펴보겠습니다. 데이터의 양이 적을 때는 문제가 되지 않지만 많을 경우 테이블 내에서 수많은 트랜잭션이 발생합니다. 이는 데이터베이스의 성능 저하에 직결되기 때문에 반드시 고려해야 하는 문제입니다. 일반적으로 많은 트랜잭션 때문에 저하되는 테이블 구조는 수평/수직 분할 설계를 통해 예방할 수 있습니다.



수평 분할은 행 단위로 요소를 분할하여 디스크의 입/출력 비용을 감소시키는 방법입니다.

- 1년 치 데이터가 대용량인 경우 월별로 분할하여 저장하면 특정 월을 조회하는 경우 전체 데이터를 조회하지 않아도 되기 때문에 성능이 향상됩니다.

수직 분할은 칼럼 단위로 요소를 분할하여 디스크의 입/출력 비용을 감소시키는 방법입니다.

- 고객의 생년월일, 주소 등의 개인정보와 취미/특기 등의 기타 정보를 별도로 저장합니다.

💡 ❕ **참고 - 데이터베이스의 입출력(I/O)**

- 데이터베이스의 입출력은 단순하게 데이터를 입력/출력 하는 것뿐만 아니라 데이터를 조작하는 모든 행위를 의미합니다. SQL문을 사용해서 작업하는 모든 행위는 데이터베이스 입출력이라고 볼 수 있습니다.
- 테이블 내의 모든 행은 블록(Block) 단위로 디스크에 저장됩니다.
- Oracle DBMS 기준 1개의 블록은 8,192바이트이며 하나의 블록마다 8,192 바이트를 저장하고 이런 블록이 모여서 테이블의 데이터가 됩니다.
- 칼럼이 많아지면 하나의 행을 저장할 때 물리적인 디스크에 여러 블록에 걸쳐 데이터가 저장될 가능성이 높아진다. 이런 경우 하나의 행을 읽더라도 여러 개의 블록을 읽는 것이 좋습니다.

수평/수직 분할을 통해서 트랜잭션에서 발생할 수 있는 데이터베이스 성능 저하를 예방한다고 하더라도 대용량 데이터를 다루는 과정에서 성능 저하는 일어날 수 있습니다. 대용량 데이터에서 발생할 수 있는 대표적인 성능 저하 현상은 로우 체이닝과 로우 마이그레이션이 있습니다.

- 성능 저하 현상

현상	설명
로우 체이닝 (Row Chaining)	- 행 데이터가 너무 길어서 데이터 블록 하나에 데이터가 모두 저장되지 않고 2개 이상의 블록에 걸쳐서 하나의 행이 저장된 형태입니다. - 하나의 행을 읽을 때 2개 이상의 데이터 블록을 읽게 되는데, 절대적으로 읽어야 하는 데이터가 증가하기 때문에 성능 저하에 직접적인 영향을 주게 됩니다.
로우 마이그레이션 (Row Migration)	- 데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에서 저장하지 못하고, 다른 블록의 빈 공간을 찾아서 저장하는 방식으로 처리됩니다. - 해당 현상이 일어나면서 하나의 행을 읽을 때 2개 이상의 데이터 블록을 읽게 됩니다. 로우 체이닝 현상과 마찬가지로 절대적으로 읽어야 하는 데이터 블록의 수가 늘어나기 때문에 성능이 저하됩니다.

로우 체이닝과 로우 마이그레이션이 발생하면 데이터베이스의 블록에 많은 데이터가 저장되고 데이터 조회 과정에서 블록 입/출력 횟수가 증가하게 됩니다. 위에서 살펴본 것처럼 디스크 입/출력 가능성이 증가하게 되는데, 이는 많은 양의 데이터를 조회하는 경우에 데이터를 메모리 내에서 조회하지 못하고 디스크에서 읽게 되기 때문입니다. 고비용의 작업이기 때문에 DBMS의 성능 저하를 유발할 수 있습니다.

💡 ❕ **참고 - 데이터베이스 파티셔닝(Partitioning)**

- 서비스의 규모가 커지고 그에 따라 저장하여 관리하는 데이터가 많아지면서 기존 시스템으로는 관리하기 어려운 상황이 발생했습니다.
- VLDB(Very Large DBMS)와 같이 하나의 DBMS에 너무 많은 테이블이 저장되면서 성능과 용량 측면에서 많은 이슈가 발생하였고 이런 이슈를 해결하기 위한 방법으로 테이블을 '나누는(partition)'하여 작은 단위로 쪼개 관리하는 '파티셔닝(partitioning)'기법이 발생하게 되었습니다.
- 파티셔닝은 논리적인 데이터를 다수의 Entity로 쪼개는 행위를 의미하는데, 큰 테이블이나 인덱스를 관리하기 쉬운 partition이라는 작은 단위로 물리적인 분할을 하는 것을 의미합니다.
- 테이블을 물리적으로 분할하지만 데이터베이스에 접근할 때는 분할된 테이블로 인식하지 못한다는 특징이 있습니다.

▼ 2) 한 테이블에 많은 수의 칼럼이 있는 경우의 성능 향상 (테이블 수직 분할)

아래 테이블은 도서 정보를 관리하기 위한 테이블입니다. 도서 정보 테이블은 관리해야 하는 데이터의 수가 많기 때문에 칼럼의 수가 매우 많습니다. 칼럼이 많다는 것은 신경 써서 관리해야 하는 정보가 많음을 의미합니다.

- 도서정보

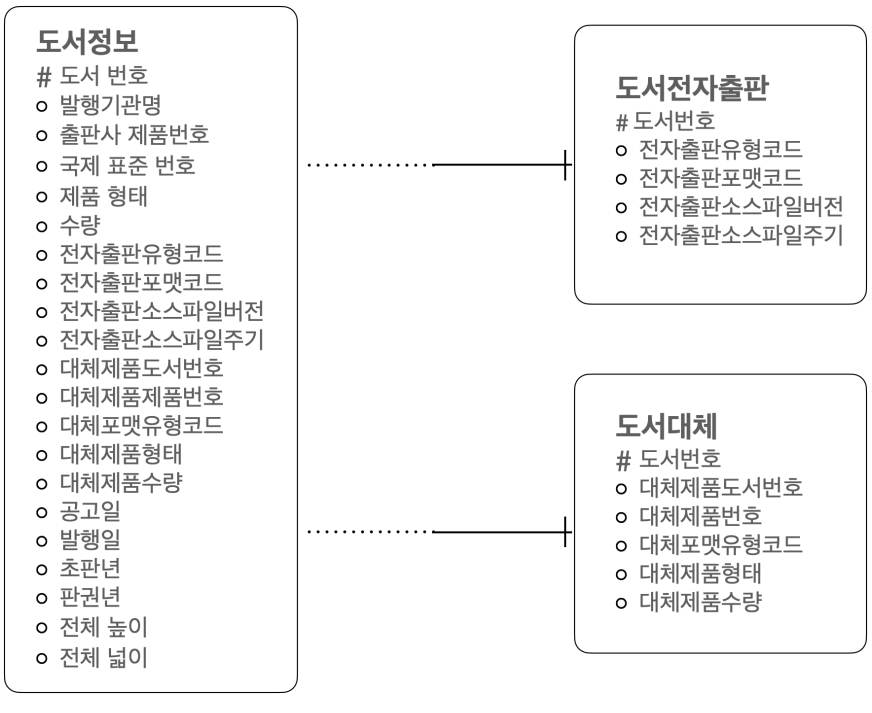
칼럼명	비고
도서 번호	PK
발행기관명	
출판사 제품번호	
국제 표준 번호	
제품 형태	
수량	
전자출판유형코드	
전자출판포맷코드	
전자출판소스파일버전	
전자출판소스파일주기	
대체제품도서번호	
대체제품제품번호	
대체포맷유형코드	
대체제품형태	
대체제품수량	
공고일	
발행일	
초판년	
판권년	
전체 높이	
전체 넓이	

만약 위의 테이블 구조에서 아래와 같은 쿼리 문을 자주 사용한다고 가정해봅시다.

```
SELECT
  발생기관명,
  수량,
  공고일,
  발행일
FROM 도서정보
WHERE 초판년 = '2002';
```

먼저 SQL 문이 실행될 때마다 **도서 정보** 테이블에 존재하는 모든 칼럼을 읽습니다. 이때 읽혀진 데이터는 모두 블록에 입력됩니다. 그 중에서 특정한 칼럼의 데이터만 가져오기 때문에 조회 대상이 아닌 칼럼을 버려지게 되어 불필요한 블록의 입/출력이 많아지게 됩니다. 이러한 과정이 빈번하게 발생한다는 것은 자연스럽게 디스크의 입/출력에도 영향을 주게 되는 걸 의미하며 결과적으로 DBMS의 전체 성능에 영향을 미치게 됩니다.

이런 경우는 '수직 분할'을 통해 문제 해결이 가능합니다. 아래 표는 도서 정보 테이블을 수직 분할 한 모습입니다.



- 도서정보

칼럼명	비고
도서 번호	PK
발행기관명	
출판사 제품번호	
국제 표준 번호	
제품 형태	
수량	
전자출판유형코드	
전자출판포맷코드	
전자출판소스파일버전	
전자출판소스파일주기	
대체제품도서번호	
대체제품제품번호	
대체포맷유형코드	
대체제품형태	
대체제품수량	
공고일	
발행일	
초판년	
판권년	
전체 높이	

- 도서전자 출판 (도서 정보 테이블과 1:1 관계)

칼럼명	비고
도서번호	PK
전자출판유형코드	
전자출판포맷코드	
전자출판소스파일버전	
전자출판소스파일주기	

- 도서대체 (도서 정보 테이블과 1:1 관계)

칼럼명	비고
도서번호	PK

칼럼명	비고
대체제품도서번호	
대체제품번호	
대체포맷유형코드	
대체제품형태	
대체제품수량	

테이블을 분할하지 않은 경우에 특정한 칼럼 데이터 조회를 위해서 모든 칼럼의 데이터를 조회해야 했습니다. 이제는 대체도서에 대한 정보를 조회하고 싶으면 도서대체 테이블에서 조회하면 되고 전자출판과 관련된 정보를 조회하고 싶으면 전자출판 테이블을 통해 조회하면 됩니다. 트랜잭션이 전자출판 및 대체제품에 종속적으로 발생하게 되면서 읽어야 하는 칼럼의 수가 현저히 줄어들게 됩니다. 이는 1개 행을 읽을 때 읽어야 하는 칼럼의 수가 줄어든다는 것이고 필요한 데이터 블록의 수가 줄어드는 결과로 이어집니다. 도서 정보 테이블을 조회하는 경우에도 디스크 입/출력이 줄어 성능을 향상시킬 수 있습니다.

▼ 3) 대량 데이터 저장 및 처리가 성능에 미치는 영향 (테이블 수평 분할)

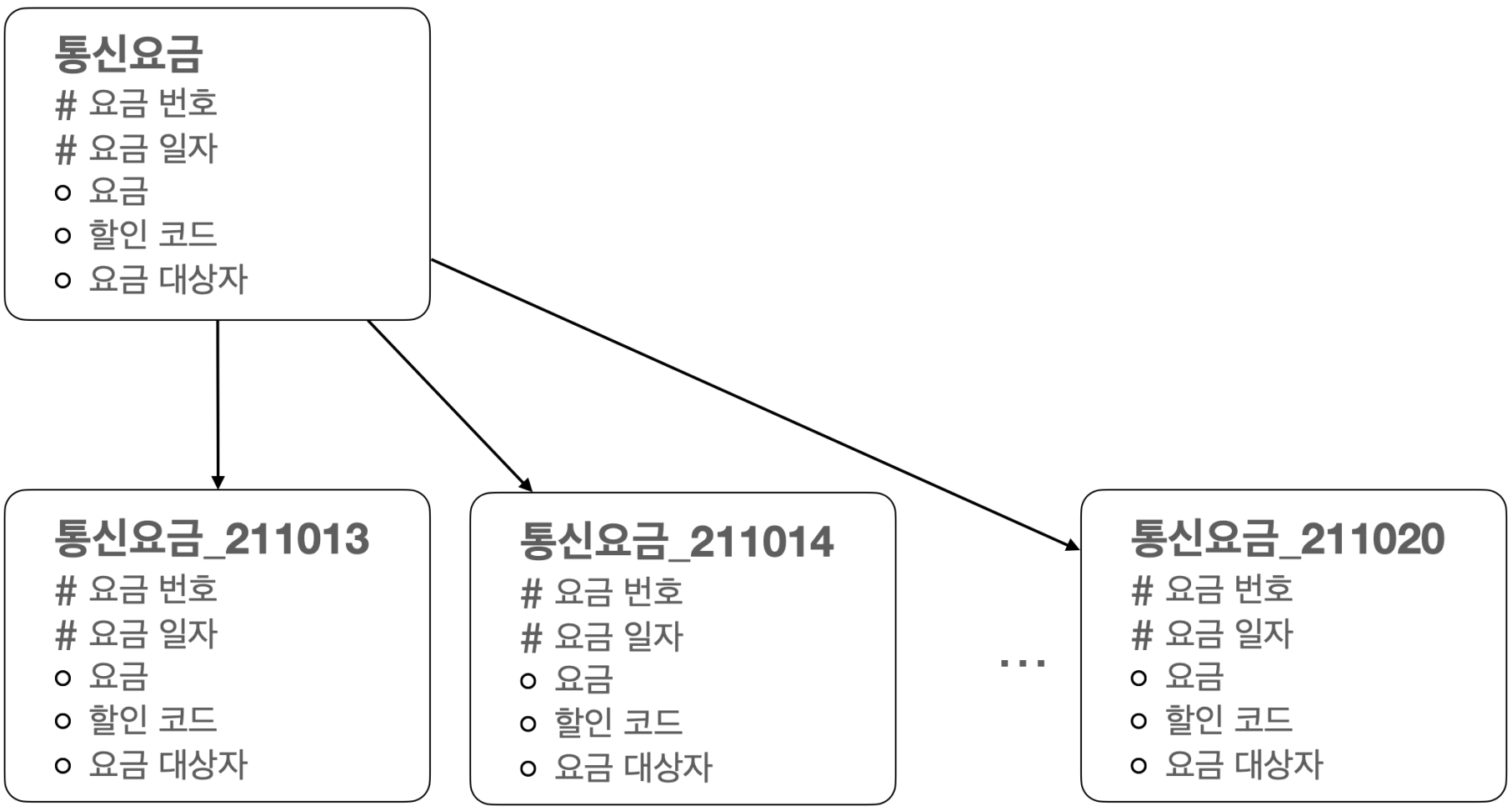
대형 통신사의 경우 한 달에 수천만 명의 고객이 통신 요금을 납부합니다. 납부하는 데이터는 모두 데이터베이스 테이블에 저장됩니다. 수백, 수천 건이 아닌 수천만 건의 데이터를 매번 저장하여 조회하는 과정은 데이터베이스의 성능에 매우 큰 영향을 주게 됩니다. 칼럼과 다르게 테이블의 저장 건수(행)가 많은 경우, 수평 분할을 통해 성능을 향상시킬 수 있습니다.

☑️ 범위 분할(RANGE PARTITION)

수평 분할은 파티셔닝이라고 하며 특정 기간을 중심으로 분할하는 것을 범위 파티셔닝이라고합니다.

통신요금 테이블에 PK는 요금일자 + 요금번호의 복합키로 구성되어 있습니다. 요금 테이블의 특성상 항상 월 단위로 데이터 처리를 하는 경우가 많기 때문에 PK 구성 칼럼을 이용하여 12개월 분량의 파티션 테이블을 생성합니다.

이렇게 할 경우 데이터의 보관 주기에 따라 테이블의 데이터를 쉽게 지울 수 있기 때문에 보관 주기에 따른 테이블 관리가 용이합니다. 그리고 보관 주기에 따른 테이블 별 조회가 가능하기 때문에 동일한 SQL문이라도 전체를 조회하는 경우보다 성능이 좋다고 볼 수 있습니다.



- 통신요금

칼럼명	비고
요금 번호	PK
요금 일자	PK
요금	

칼럼명	비고
할인 코드	
요금 대상자	

- 통신요금_211013

칼럼명	비고
요금번호	PK
요금일자	PK
요금	
할인코드	
요금 대상자	

- 통신요금__211014

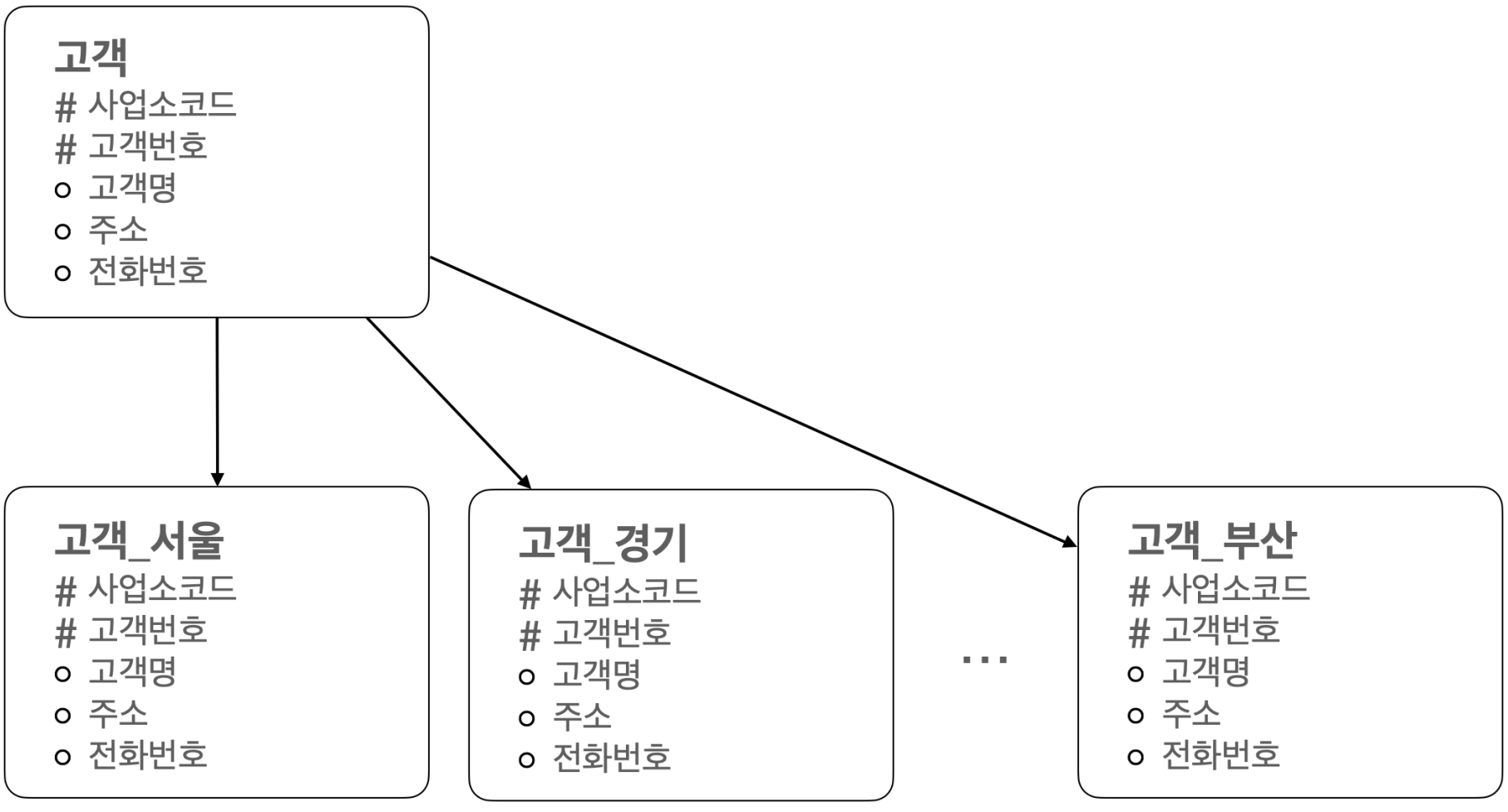
칼럼명	비고
요금번호	PK
요금일자	PK
요금	
할인코드	
요금 대상자	

- 통신요금__211015

칼럼명	비고
요금번호	PK
요금일자	PK
요금	
할인코드	
요금 대상자	

☑️ 목록 분할(LIST PARTITION)

리스트 파티션의 경우 행의 수가 매우 많습니다. 그럴 때 아래처럼 테이블을 분리할 경우에 경기권 고객만을 대상으로 데이터를 조회할 경우에는 '고객_경기' 테이블의 파티션만 조회하면 되기 때문에 SQL문의 성능이 향상될 수 있습니다.



- 고객

칼럼명	비고
사업소코드	PK
고객번호	PK
고객명	
주소	
전화번호	

- 고객_서울

칼럼명	비고
사업소코드	PK
고객번호	PK
고객명	
주소	
전화번호	

- 고객_경기

칼럼명	비고
사업소코드	PK
고객번호	PK
고객명	
주소	
전화번호	

- 고객_강원

칼럼명	비고
사업소코드	PK
고객번호	PK
고객명	
주소	
전화번호	

☑️ 해시 분할(HASH PARTITION)

해시 파티셔닝은 지정된 HASH 조건에 따라 해싱 알고리즘을 적용하여 테이블을 분리합니다. 주문 번호나 주민번호 같은 UNIQUE 칼럼을 기준으로 테이블을 분할하는 기법입니다. 데이터의 관리보다는 성능 향상에 그 목적이 있습니다.

장점

- 해싱 알고리즘에 의해 분리되어 데이터가 입력되기 때문에 기존 1개의 테이블에만 데이터를 입력하는 방식보다 부하가 줄어듭니다.
- 특정 파티션에 데이터가 집중될 가능성이 있는 범위 분할의 단점을 보완할 수 있습니다.
- 데이터 처리가 많아지는 경우에 경합을 막을 수 있습니다.

단점

- 데이터 보관 주기에 따라 쉽게 삭제하는 기능을 제공하기 어렵습니다.
- 설계자 및 데이터 입력자는 특정 데이터가 어떤 파티션에 저장되는지 정확하게 예측하기 어렵습니다.

☑️ 합성 분할(COMPOSITE PARTITIONING)

- 위에서 살펴본 3개의 분할 방식을 섞는 방법을 의미하는데, 범위 분할로 분할 이후 다시 해시 함수를 적용하여 분할하는 방식등을 말합니다.

▼ 4) 테이블에 대한 수평/수직 분할 절차

테이블을 수평 / 수직 분할하는 과정은 아래와 같습니다.

1. 데이터베이스 모델링을 진행합니다.
2. 데이터베이스 테이블의 용량 산정합니다.
3. 데이터 처리 과정에서 트랜잭션 처리 패턴 분석합니다.
4. 데이터 처리의 과정이 칼럼과 로우 중 어디에 집중되는지 분석하고 집중된 부분의 테이블을 파티셔닝합니다.

▼ 연습문제

☑️ 문제 1

Q. 문제	성능 저하 현상에 대해 바르게 짝 지어진 것은?
A. (1)	로우 체이닝 - 행 데이터가 데이터 블록 하나에 모두 저장된 형태
A. (2)	로우 체이닝 - 데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에서 저장하지 못하고, 다른 블록의 빈 공간을 찾아서 저장하는 방식
A. (3)	로우 마이그레이션 - 절대적으로 읽어야 하는 데이터 블록의 수가 줄어들기 때문에 성능이 저하되는 현상
A. (4)	로우 마이그레이션 - 데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에서 저장하지 못하고, 다른 블록의 빈 공간을 찾아서 저장하는 방식

▼ 정답

(4)

- (1) 로우체이닝 : 행 데이터가 너무 길어서 **데이터 블록 하나에 데이터가 모두 저장되지 않고** 2개 이상의 블록에 걸쳐서 하나의 행이 저장된 형태
- (2) 해당 내용은 로우 체이닝이아닌 로우 그레이션에 대한 설명
  - 하나의 행을 읽을 때 2개 이상의 데이터 블록을 읽게 되는데, 절대적으로 읽어야 하는 데이터가 증가하기 때문에 성능 저하에 직접적인 영향을 주게 된다
- (3) 로우 체이닝 현상과 마찬가지로 절대적으로 읽어야 하는 **데이터 블록의 수가 늘어나기 때문에** 성능이 저하된다

☑️ 문제 2

Q. 문제	파티션 기법 중 2개 이상의 기법을 사용하는 것은 무엇인가?
A. (1)	범위 분할(RANGE PARTITION)
A. (2)	목록 분할(LIST PARTITION)
A. (3)	해시 분할(HASH PARTITION)
A. (4)	합성 분할(COMPOSITE PARTITIONING)

▼ 정답

(4)

☑️ 문제 3

Q. 문제	해시분할의 장점으로 옳지 않은 것은?
A. (1)	기존 1개의 테이블에만 데이터를 입력하는 방식보다 부하가 줄어든다
A. (2)	데이터 보관 주기에 따라 쉽게 삭제하는 기능을 제공한다
A. (3)	데이터 처리가 많아지는 경우에 경합을 막을 수 있다
A. (4)	데이터 입력 시 경합에 의한 성능 부하를 해소하기 위해 사용한다

▼ 정답

(2) 데이터 보관 주기에 따라 쉽게 삭제하는 기능을 제공하기 어렵다

02. 데이터베이스의 구조와 성능

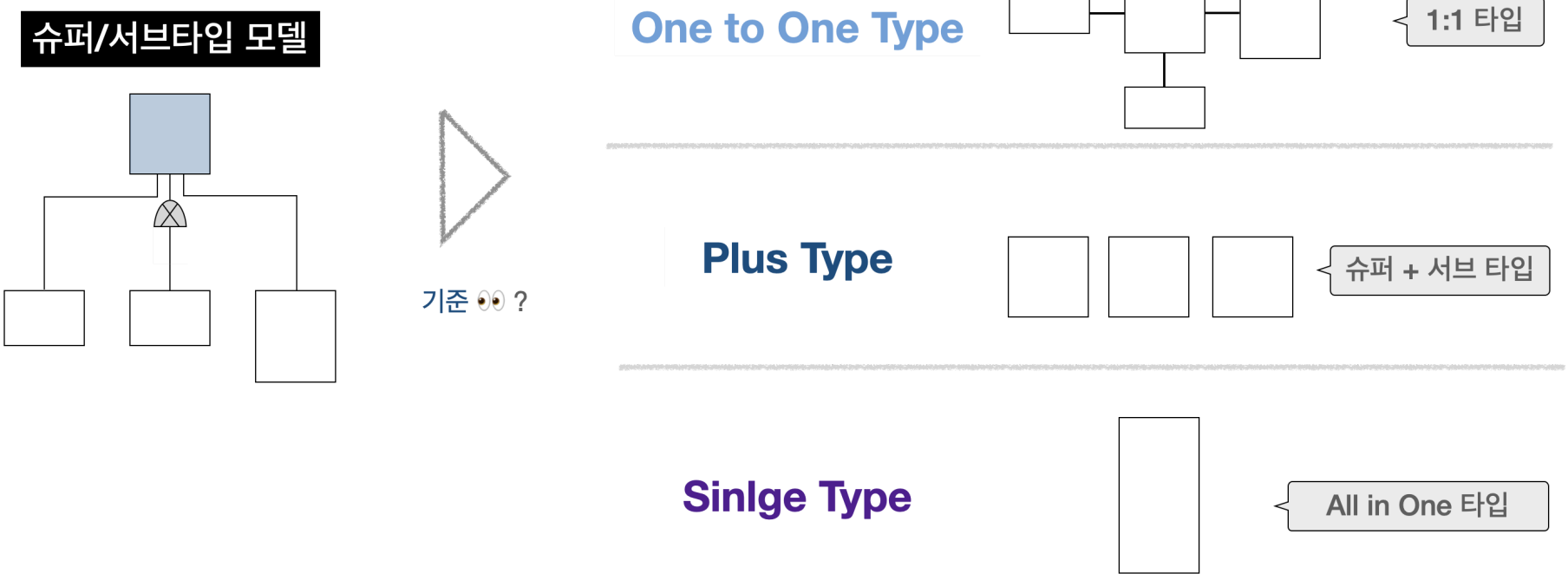
> ✔️ **슈퍼 타입/서브 타입의 구조와 성능에 대해 학습합니다**

▼ 1) 슈퍼타입 / 서브타입 모델의 성능 고려방법

☑️ 슈퍼타입/서브타입 기본

슈퍼타입 / 서브타입 데이터 모델은 데이터 모델링을 할 때 자주 쓰이는 기법입니다. 업무를 구성하는 데이터의 특징을 공통점과 차이점으로 나누어 표현 가능하기 때문에 직관적인 모델링이 가능합니다.

슈퍼타입은 공통적인 속성이고 서브타입은 자신만의 속성을 의미합니다. 공통적인 부분을 슈퍼 타입 엔터티로 만들고 그것으로부터 상속받아 다른 엔터티와 차이가 있는 속성에 대해서는 별도의 서브 타입 엔터티로 구분하는 방식입니다. 슈퍼타입과 서브타입 모델의 변환 과정을 도식화하여 살펴보면 다음과 같습니다.



☑️ 슈퍼타입 / 서브타입 모델 변환 방법

방법	설명
슈퍼 타입(Single / All in One)	- 슈퍼/서브 타입 모델을 하나의 테이블로 변환합니다. - 고객 테이블을 싱글 테이블로 구성합니다.
서브 타입(Plus / Super + Sub)	- 슈퍼타입과 서브타입 테이블로 변환합니다. - 도출된 각 서브 타입에는 변환 전에 슈퍼 엔터티에 있던 칼럼을 공통적으로 가지고 있습니다.
개별 타입(OneToOne + 1:1)	- 슈퍼/서브 타입을 슈퍼 타입과 서브 타입의 각 개별 테이블로 변환합니다. - 슈퍼/서브 테이블 모두를 생성합니다.

☑️ 슈퍼타입 / 서브타입 데이터 모델 변환 타입 비교

구분	슈퍼 타입	서브 타입	개별 타입
특징	하나의 테이블	각각의 서브타입 테이블	슈퍼 서브 각각의 테이블
확장성	나쁨	보통	좋음
조인성능	우수함	나쁨	나쁨
I/O 성능	나쁨	좋음	좋음
관리 용이성	좋음	좋지 않음	좋지 않음

☑️ 슈퍼타입 / 서브타입 모델 변환의 중요성

1. 트랜잭션은 항상 일괄적으로 처리됩니다.
    - 테이블은 개별적으로 유지되기 때문에 UNION 연산에 의해 성능 저하의 가능성이 있습니다.
2. 트랜잭션은 항상 서브타입을 개별로 처리됩니다.
    - 테이블을 하나로 통합되어 있어 불필요하게 많은 데이터를 처리하는 과정에서 성능 저하의 가능성이 있습니다.
3. 트랜잭션은 항상 슈퍼타입과 서브타입을 공통으로 처리됩니다.
    - 만약 개별로 유지되거나 하나의 테이블로 집약되어 있어 성능 저하를 보일 수 있습니다.

트랜잭션이 얼마만큼 빈번하게 처리되는지에 따라 테이블을 설계해야 성능 저하에 대한 부분을 최소화시킬 수 있습니다. 슈퍼타입과 서브타입 성능을 고려한 물리적인 데이터 모델로 변환하는 기준은 데이터가 얼마나 많은지와 테이블에서 발생하는 트랜잭션의 유형에 따라서 결정됩니다.

▼ 2) 인덱스 특성을 고려한 PK / FK 데이터베이스 성능 향상

☑️ PK 순서와 성능 사이의 관계

테이블에는 기본키(PK)가 존재하고 PK는 단일 칼럼으로 구성되거나(단일 PK) 2개 이상의 칼럼으로 구성(복합 PK)될 수 있습니다. 복합 PK의 경우 데이터의 성능 향상을 위해 테이블에 발생되는 트랜잭션 조회 패턴에 따라 칼럼의 순서를 조정할 필요가 있습니다. PK 칼럼이 테이블의 어느 곳에 위치해 있는지에 따라서 SQL 쿼리문의 성능이 달라질 수 있기 때문입니다. 이 부분에 대해 조금 더 구체적으로 알아보도록 하겠습니다.

- **PK 순서 지정 실수로 인한 성능 저하 예시1**

칼럼명	비고
수험번호	복합 PK
연도	복합 PK
학기	복합 PK
대학원구분코드	
학위구분코드	
등기부상주소	

위 테이블에서 PK는 복합키로 구성되어 있으며 '수험번호 + 년도 + 학기' 칼럼의 조합이 기본키로 설정되어 있습니다. 이런 상황에서 아래와 같은 쿼리문이 **자주 호출된다고** 가정해 봅시다.

```
SELECT COUNT(입시번호)
FROM 입시정보
WHERE 연도 = '2021' AND 학기 = '2';
```

수험 번호 칼럼이 WHERE 절에 포함되지 않았기 때문에 테이블에 대한 전체 조회가 발생하게 되어 데이터의 총개수만큼의 행 조회가 발생하게 됩니다. 당연하게도 성능이 저하됩니다. 이러한 상황에서 복합 PK에 대한 인덱스 스캔을 유도할 수 있습니다. 칼럼의 순서만 변경하면 쉽게 달성 가능합니다.

아래 테이블과 같이 연도와 학기 칼럼을 위로 올린 다음 위에서 수행한 SQL문을 그대로 넣고 실행하면, 이번에는 2가지 칼럼을 토대로 조회가 가능하기 때문에 모든 행을 스캔하지 않아도 됩니다. 따라서 자연스럽게 성능 조회를 이룰 수 있습니다.

<입시 정보>

칼럼명	비고
연도	복합 PK
학기	복합 PK
수험번호	복합 PK
대학원구분코드	
학위구분코드	
등기부상주소	

- **PK 순서 지정 실수로 인한 성능 저하 예시2**

<현금 출금기 정보>

칼럼명	비고
거래일자	복합PK
사무소코드	복합PK
출금기번호	복합PK
명세표번호	복합PK
건수	
금액	

위 테이블에서는 첫 예시처럼 PK가 복합키로 구성되어 있습니다. 복합키는 '거래일자 + 사무소코드 + 출금기번호 + 명세표번호' 이렇게 4가지의 속성이 하나로 묶여 PK를 구성합니다. 이러한 테이블에서 아래와 같은 SQL문을 수행한다고 가정해 봅시다.

```
SELECT
  건수, 금액
FROM '현금출금기정보'
WHERE 거래일자 BETWEEN '20211016' AND '20211020' AND 사무실코드 = 'BC2011';
```

첫 번째 예시와 비교해 봤을 때 거래일자와 사무실코드는 이미 복합키의 상단에 존재해 테이블을 스캔하는 과정에서 전체를 보지 않아도 되는 점에서는 성능적인 부분의 문제는 없습니다. 하지만 이번에는 조건 자체가 단순하게 특정한 값 하나만을 찾는 것이 아니라 BETWEEN AND 연산자를 사용하여 범위로 스캔하기 때문에 문제가 생깁니다. 아래와 같이 테이블 순서를 변경해봅시다.

<현금 출금기 정보>

칼럼명	비고
사무소코드	복합PK
거래일자	복합PK
출금기번호	복합PK
명세표번호	복합PK
건수	
금액	

이렇게 순서를 바꾸게 되면 첫 번째와 다르게 거래일자를 먼저 보고 해당 일자에 속하는 모든 범위를 스캔하지 않고 사무실 코드가 BC2011인 데이터에서 범위를 먼저 스캔합니다. 따라서 찾아야 하는 데이터의 크기가 현저하게 줄어들기 때문에 복합키의 순서만 변경하고도 충분한 성능 향상을 이룰 수 있습니다.

☑️ FK와 테이블 인덱스 구성 사이의 관계

논리 데이터 모델 상으로 관계에 의한 외래키(FK) 제약이 걸린 경우 실제 물리 데이터베이스에 대한 적용 여부는 물리 데이터베이스가 어떻게 설계되어 있는지에 따라 다릅니다. 테이블 사이의 관계가 있지만 FK 제약 조건을 생성하지 않는 경우도 있고, FK 제약 조건을 생성하였음에도 FK 칼럼의 인덱스를 생성하지 않은 경우가 있습니다. 사실 물리적인 외래키(FK)의 생성 여부와 상관없이 논리/물리 FK 제약 조건은 외래키 칼럼에 대해 인덱스를 생성하는 것이 성능 상의 우위를 가질 확률이 높습니다.

아래와 같은 테이블 정보가 있다고 가정해 봅시다. 수강 정보는 기본적으로 학사 정보 테이블의 데이터와 연결되는 부분이 많기 때문에 테이블 조인이 많이 발생하게 됩니다. 학사 정보 테이블에서 특정한 데이터를 읽고(행) 학사기준번호 PK를 기준으로 수강 정보 테이블에서 맞는 결과를 조회합니다.

- 학사 정보

칼럼명	비고
학사기준번호	PK, ID
연도	
학기	
특이사항	

- 수강 정보

칼럼명	비고
강의번호	ID
학번	
학사기준번호	FK
성명	
연락처	
등록연도	
감면코드	

이때 수강 정보 테이블에 학사기준번호 칼럼에 대한 인덱스가 존재하지 않을 경우 조인을 수행하는 과정에서 수강신청 테이블에 대한 데이터를 테이블 전체의 row 만큼 스캔하기 때문에 필연적으로 성능 저하가 일어납니다. 이런 문제점을 해결하기 위해서 수강 정보 테이블의 학사기준번호 칼럼으로 하는 인덱스를 미리 만들어 놓으면 학사 정보 테이블에서 조인을 하는 과정에서 성능 저하를 막을 수 있습니다.

▼ 연습문제

☑️ 문제 1

Q. 문제	슈퍼 타입과 서브 타입의 관계에서 변환되는 테이블 수가 가장 적은 것은?
A. (1)	Single Type
A. (2)	Plus Type
A. (3)	OneToOne Type
A. (4)	All in One Type

▼ 정답

(1) Single Type은 슈퍼/서브 타입 모델을 하나의 테이블로 변환한다

☑️ 문제 2

Q. 문제	슈퍼타입/서브타입의 설명으로 옳지 않은 것은?
A. (1)	슈퍼타입/서브타입 데이터 모델은 데이터 모델링을 할 때 자주 쓰이는 기법이다
A. (2)	슈퍼타입은 공통적인 속성이고 서브타입은 자신만의 속성을 의미한다
A. (3)	공통적인 부분을 슈퍼 타입 엔터티로 만든다
A. (4)	업무를 구성하는 데이터의 특징을 장점과 단점으로 나누어 표현 가능하기 때문에 직관적인 모델링이 가능하다

▼ 정답

(4) 업무를 구성하는 데이터의 특징을 **공통점과 차이점**으로 나누어 표현 가능하기 때문에 직관적인 모델링이 가능합니다.

☑️ 문제 3

Q. 문제	슈퍼타입/서브타입 모델의 변환 타입에대한 설명으로 옳은 것은?
A. (1)	개별타입은 슈퍼 및 서브 타입 각각의 테이블로 변환하기 때문에 관리 용이성이 뛰어나다
A. (2)	슈퍼타입은 하나의 테이블로 통합하기 때문에 관리 용이성이 좋지 않다
A. (3)	서브타입은 서브타입별로 테이블을 유지하기 때문에 관리 용이성이 뛰어나다
A. (4)	서브타입의 확장성은 보통 수준이며 관리 용이성이 좋지 않다

▼ 정답

(4)

- (1) 개별타입은 관리 용이성이 좋지 않다
- (2) 슈퍼타입은 관리 용이성이 좋다
- (3) 서브타입은 관리 용이성이 좋지 않다

03. 분산 데이터베이스와 성능

✔️ **분산 데이터베이스의 개념과 특징에 대해 학습합니다**

▼ 1) 분산 데이터베이스의 개요

☑️ 분산 데이터베이스의 개요

분산 데이터베이스는 여러 곳에 분산되어 있는 데이터베이스 시스템을 하나의 가상 시스템으로 사용할 수 있도록 한 데이터베이스입니다. 물리적으로 동일한 위치에 여러 대의 컴퓨터로 구성된 경우 혹은 컴퓨터 네트워크에서 상호 연결된 컴퓨터 군에 분산되어 있는 경우 등이 있습니다.

논리적으로는 동일한 시스템이지만 네트워크를 통해 물리적으로 분산되어 있는 데이터의 모임을 의미합니다. 물리적으로는 분산되어 있지만 논리적으로는 여러 사람들이 사용할 수 있는 것처럼 가상의 시스템을 구축하여 데이터를 관리합니다.

분산 데이터베이스는 아래와 같은 특징을 갖는다고 볼 수 있습니다.

- 여러 곳으로 분산되어 있는 데이터베이스를 하나의 가상 시스템으로 사용할 수 있도록 합니다.
- 논리적으로 동일한 시스템에 속하지만 컴퓨터 네트워크를 통해 물리적으로 분산되어 있는 데이터들의 모임입니다.
- 물리적으로는 분산되어 있지만 논리적으로는 하나의 데이터베이스를 다루는 것처럼 사용자를 통합합니다.

☑️ 분산 데이터베이스의 투명성

분산 데이터베이스가 되기 위해서는 6가지의 투명성(Transparency)를 만족해야 합니다. 여기서 투명성은 해당 데이터베이스를 사용하는 사용자가 데이터베이스 시스템이 논리적으로 분산되어 있음을 인식하지 못하고 나만의 데이터베이스 시스템을 사용하는 것처럼 느끼게 만드는 것입니다.

종류	설명
분할 투명성	- 하나의 논리적 관계(Relation)가 여러 단편으로 분할되어 각 사본이 여러 사이트에 저장됩니다. - 이때 사용자는 분할되어 저장되는게 아니라 한 곳에 모두 위치한 것처럼 느껴야 합니다.
위치 투명성	- 위치 정보는 System Catalog에 유지되어야 합니다. - 사용자는 자신의 데이터가 어디에 위치하는지 신경 쓸 필요가 없어야하며 결과적으로 사용하려는 데이터의 저장소를 명시할 필요가 없습니다.
지역 사상 투명성	- 지역 DBMS와 물리적 DB 사이의 연결됨(mapping)을 보장합니다. - 각 지역 시스템 이름과 무관한 이름이 사용 가능합니다. - 사용자는 해당 데이터가 어떤 지역에 위치하는지 알 필요가 없습니다.
중복 투명성	- DB 객체가 여러 사이트에 중복되어 있는지 알 필요가 없습니다.
장애 투명성	- 구성요소에서 발생하는 장애에 무관한 트랜잭션의 원자성(Atomicity)을 유지합니다.
병행 투명성	- 다수 트랜잭션이 동시에 수행 시 결과의 일관성을 유지합니다. - 1단계는 Time stamp 2단계는 Locking을 이용하여 구현합니다.

☑️ 분산 데이터베이스 적용 방법

분산 데이터베이스가 모든 문제를 해결하는 만능은 아닙니다. 환경을 구축하기 위한 개발 비용 및 처리 비용이 증가하고 응답 속도 등이 원하는 만큼 개선되지 않을 수 있는 문제가 있습니다. 따라서 모든 환경에서 분산 데이터베이스를 적용하는 것이 아니라 업무 구성에 따른 아키텍처가 어떤 형태로 구성되어 있는지 등을 충분하게 고려하여 적용해야 합니다.

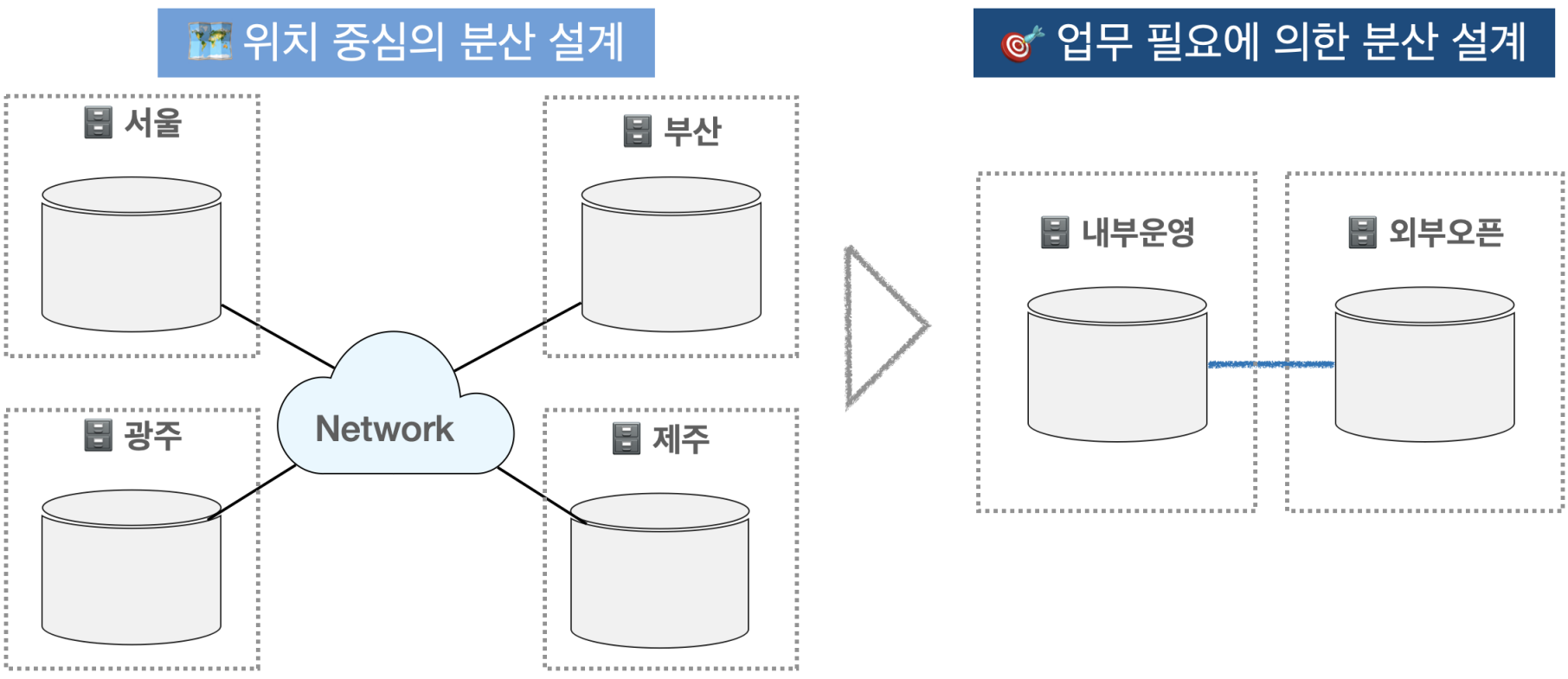
☑️ 분산 데이터베이스의 장단점

장점	단점
지역 자치성, 점증적 시스템 용량 확장	소프트웨어 개발 비용 증가

장점	단점
신뢰성 & 가용성	오류의 잠재성 증대
효용성 & 융통성	처리 비용의 증대
빠른 응답 속도 & 통신 비용 절감	설계 및 관리의 복잡성과 비용
데이터의 가용성 & 신뢰도 증가	불규칙한 응답 속도
시스템 규모의 적절한 조절	통제의 어려움
각 지역 사용자의 요구 수용 증대	데이터 무결성에 대한 위협 (전처리 필요)

▼ 2) 분산 데이터베이스의 활용 방향성 및 적용 기법

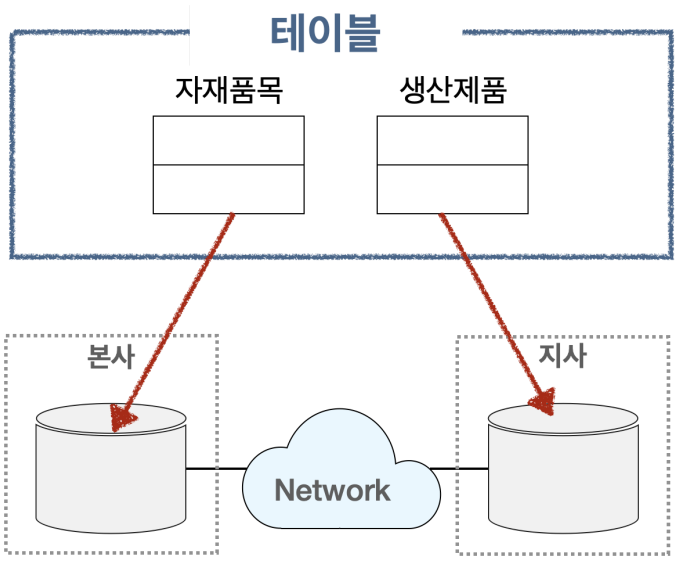
☑️ 분산 데이터베이스의 활용 방향성



과거의 분산 데이터베이스 시스템은 '위치' 중심의 설계였습니다. 데이터베이스 서버는 물리적인 지역별로 나눠져있고 서버 간의 네트워크로 연결된 구조였습니다. 최근에는 업무 특성에 맞게 내부 및 외부 데이터베이스를 나눠서 설계를 진행합니다. 통합된 데이터베이스 환경보다 훨씬 더 빠른 성능을 제공합니다. 다만, 원거리 혹은 다른 서버에 접속하면서 발생하는 네트워크 트래픽 및 혹은 트랜잭션 집중으로 인한 성능 저하가 발생할 수 있습니다.

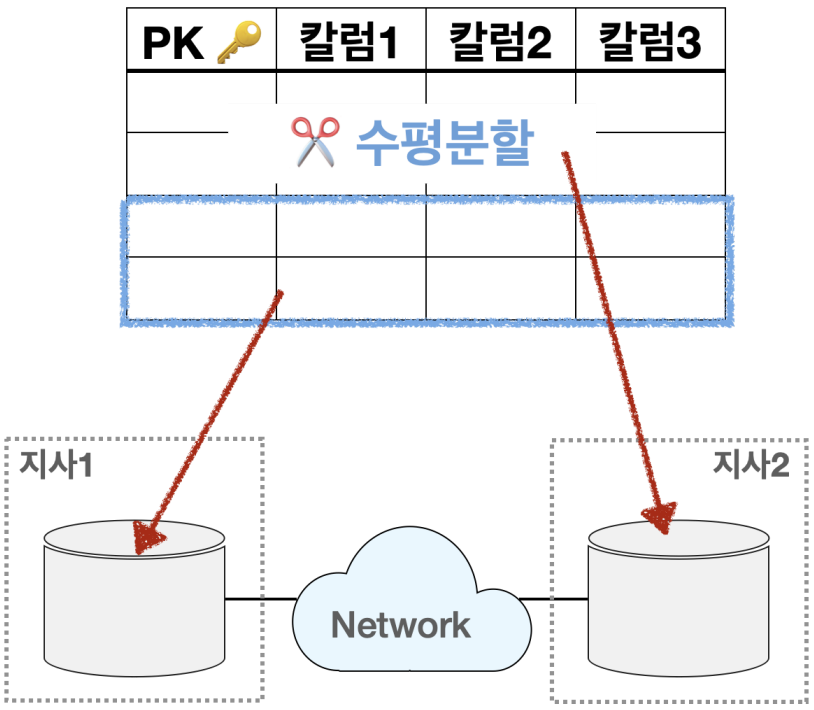
☑️ 분산 데이터베이스의 적용 기법

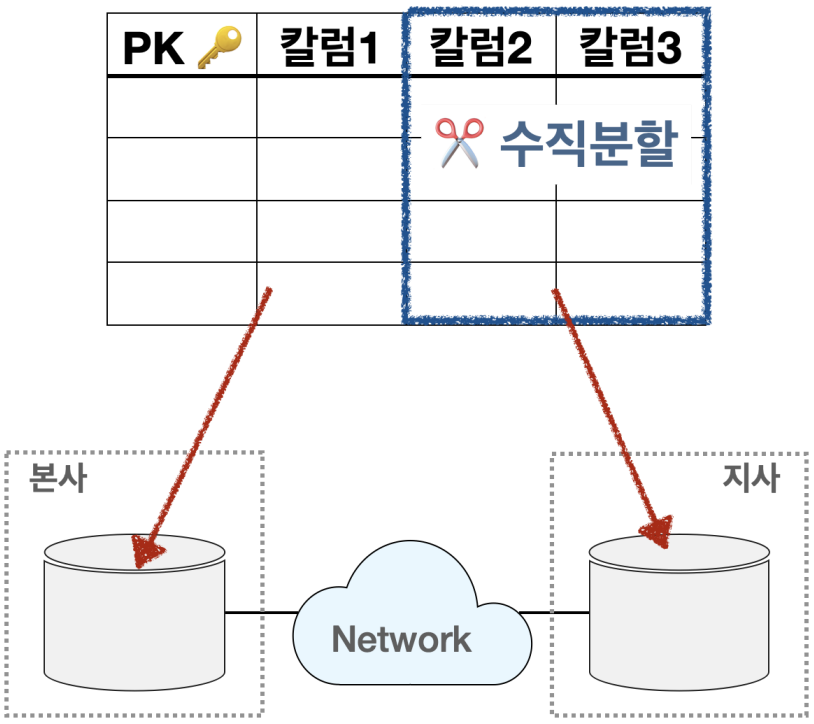
(1) 테이블 위치(Location) 분산



설계된 테이블의 위치를 다르게 위치시킵니다. 예를 들어, 자재 품목은 본사에서 구입하고 각 지사별로 자재 품목을 이용하여 제품을 구분합니다.

(2) 테이블 분할(Fragmentation) 분산





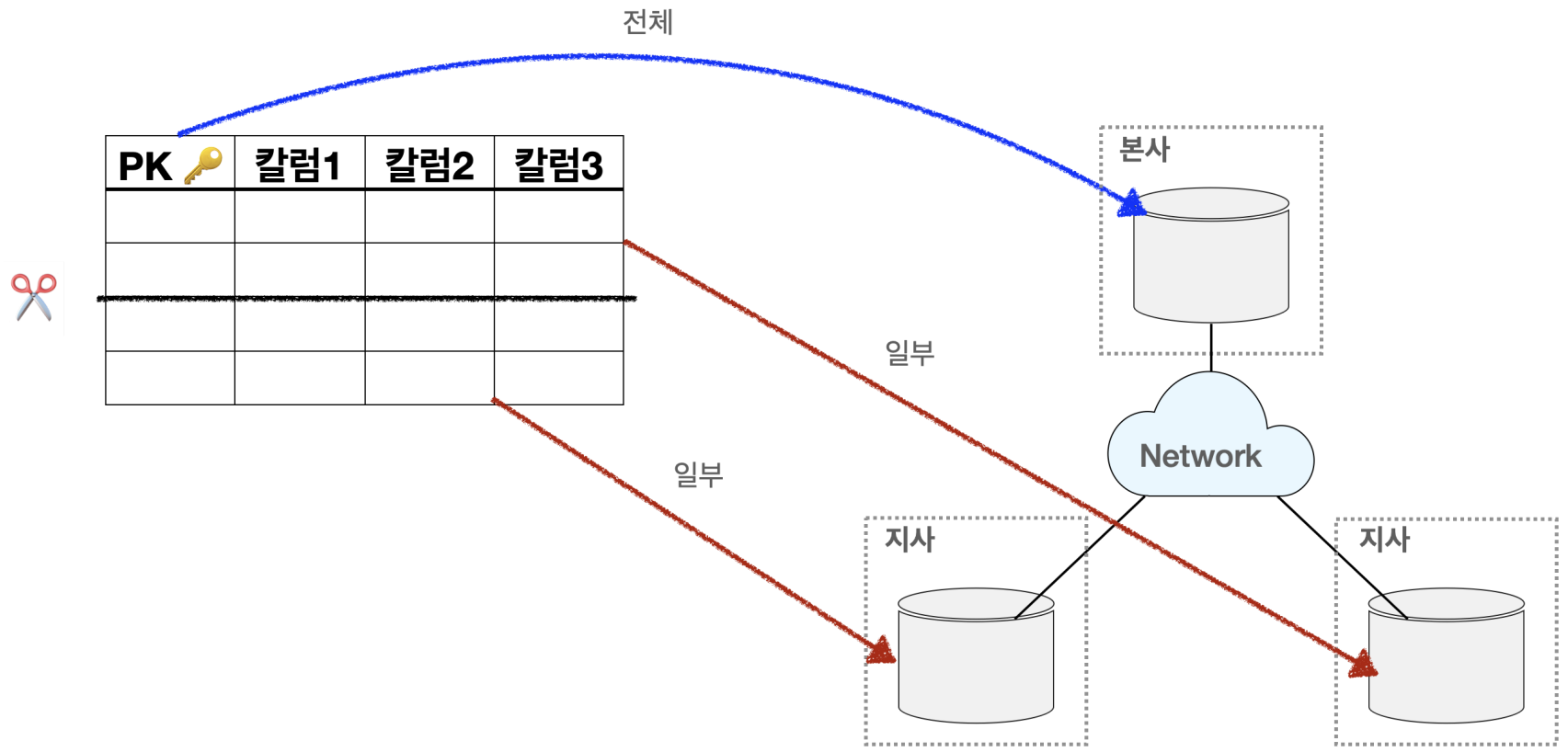
테이블 분할 분산은 단순히 위치만 다른 곳에 두는 것이 아니라 각 테이블을 쪼개서 분산하는 방법입니다. 테이블을 나누는 기준에 따라 이전 레슨에서 배웠던 수평/수직 분할로 구분할 수 있습니다.

- **수평 분할(Horizontal Fragmentation)**
  - 테이블을 특정 칼럼을 기준으로 로우(row)로 분리합니다.
  - 칼럼은 분리되지 않으며 모든 데이터는 분리되어 있는 형태로 구성됩니다.
- **수직 분할(Vertical Fragmentation)**
  - 테이블 칼럼을 기준으로 칼럼(column)으로 분리합니다.
  - 로우 단위로는 분리되지 않으며 모든 데이터는 분리되어 있는 형태로 구성됩니다.

(3) 테이블 복제(Replication) 분산

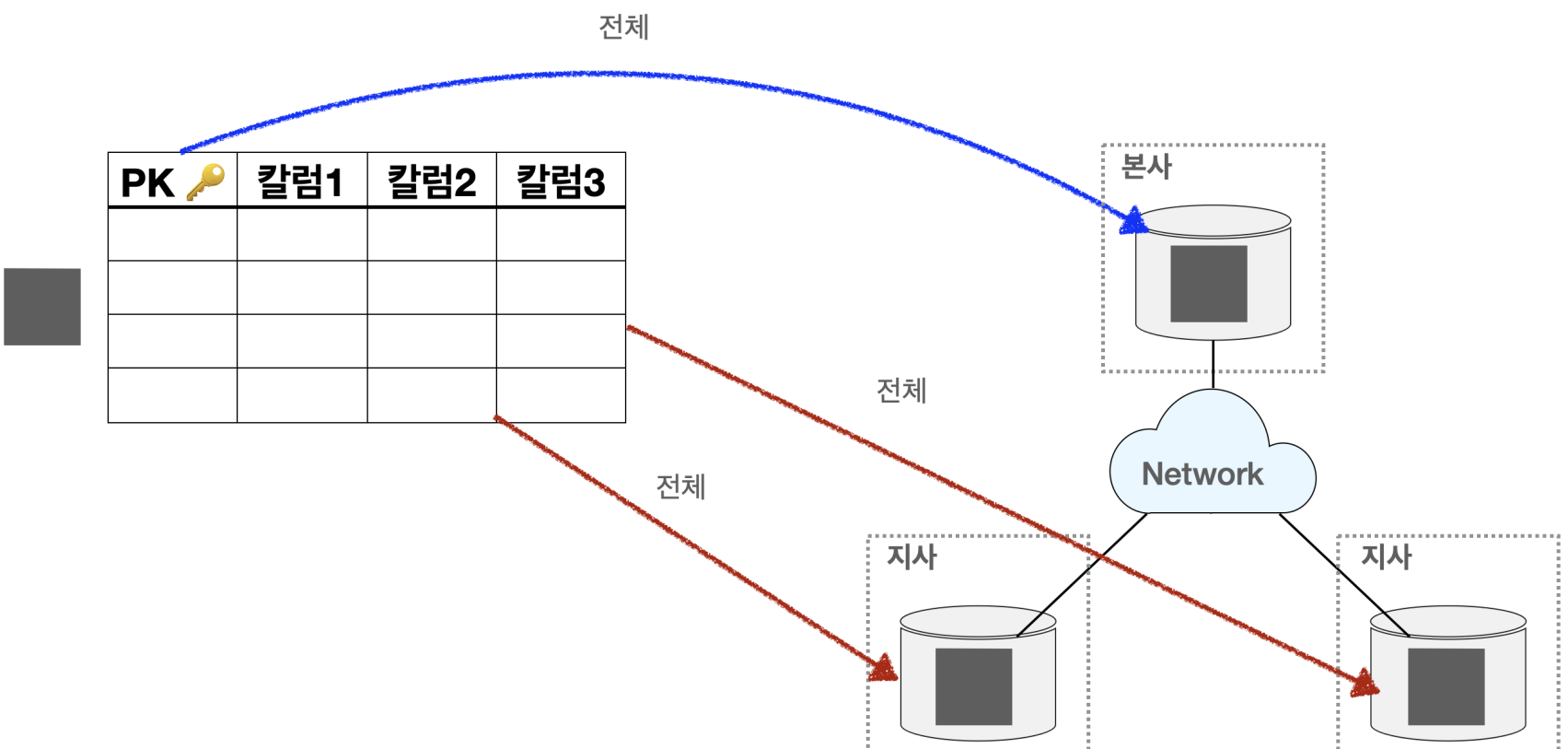
동일한 테이블을 다른 지역이나 서버에서 동시에 생성하고 관리하는 유형입니다. 크게 부분/광역 복제로 구분할 수 있습니다.

- **부분복제(Segment Replication)**
  - 통합된 테이블을 마스터 데이터베이스에 가지고 있으면서 각 지사별로는 지사에 해당하는 로우(row)를 가지고 있는 형태입니다.



- **광역복제(Broadcast Replication)**

- 통합된 테이블을 마스터 데이터베이스에 가지고 있으면서 각 지사별로는 마스터 데이터베이스와 동일한 데이터를 가지고 있는 형태입니다.



(4) 테이블 요약(Summarization) 분산

지역 / 서버 간의 데이터가 비슷하지만 서로 다른 유형으로 존재하는 경우에 해당합니다. 요약의 방식에 따라 분석요약(Rollup Summarization)과 통합요약(Consolidation Summarization)으로 구분할 수 있습니다.

- **분석요약(Rollup Summarization)**
  - 동일한 테이블 구조를 가지고 있으면서 분산되어 있는 동일한 내용의 데이터를 이용한 통합된 데이터를 산출하는 방식입니다.

    → 유저의 활동이 가장 적은 시간에 작업해서 요약하겠죠?
- **통합요약(Consolidation Summarization)**
  - 분산되어 있는 다른 내용의 데이터를 이용하여 통합된 데이터를 산출하는 방식입니다.

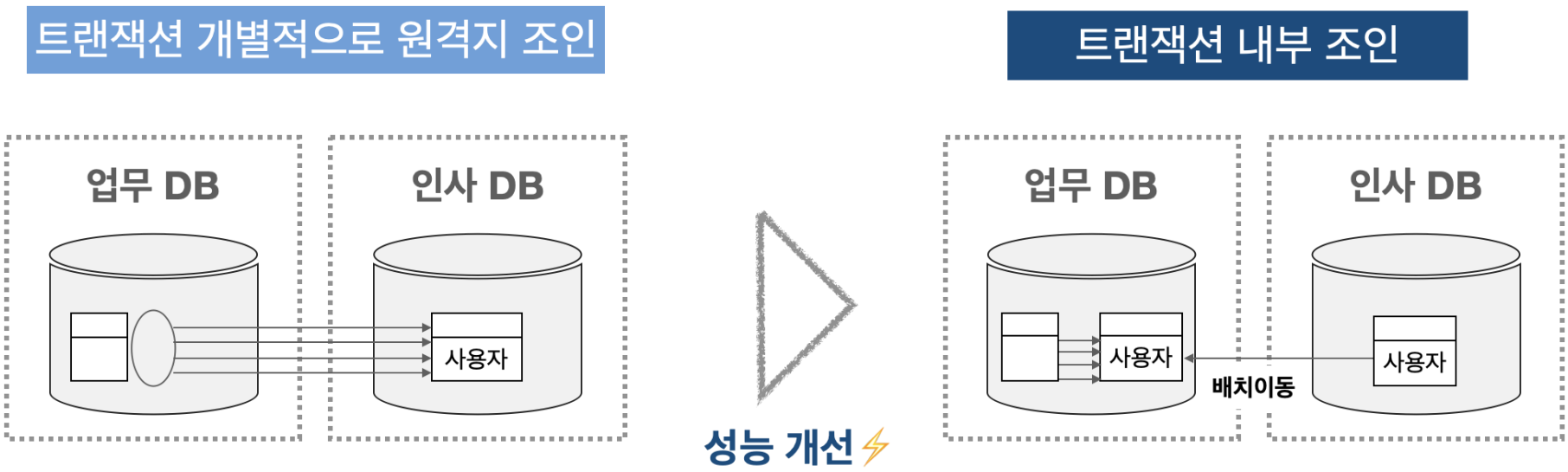
    → 중앙으로 모아서 다시 산출해주는 방식
- 분석 요약 vs 통합 요약
  - 분석 : A라는 상품에 대한 모든 판매 실적을 모든 분산 데이터베이스가 가지고 있지만
    디테일한 실적은 모두 다르며, 통합하는 데이터베이스가 전체를 산출
  - 통합 : 각 분산 데이터베이스 마다 다른 상품에 대한 데이터를 가지고 있고, 이를 통합 데이터베이스가 모아서 전체를 산출

▼ 3) 분산 데이터베이스를 활용한 성능 향상 사례

데이터베이스 프로젝트를 수행할 때 단순한 분산 환경의 원리를 이해하지 않고 데이터베이스를 설계하는 경우 성능이 저하되는 경우가 빈번하게 발생합니다.

만약 트랜잭션 개별적으로 원격 서버에서 조인을 발생시키는 경우가 있다고 생각해 봅시다. 메인 업무를 처리하는 과정에서 원격 서버에 있는 인사 데이터베이스의 사용자 정보를 원격으로 조회해야 합니다. 메인 데이터베이스의 처리 요청이 들어오는 순간마다 매번 원격 데이터베이스의 데이터를 조회해야 하기 때문에 테이블의 빈번한 조인 연산과 네트워크 트래픽 부하 등으로 인한 성능 저하가 발생할 수 있습니다.

이를 해결하기 위해 업무 특성에 따른 분산 환경 구성이 필요합니다. 오른쪽 아래와 같은 분산 환경 구축이 필요한데, 인사 데이터베이스에 있는 사용자 정보를 업무 데이터베이스로 복제하여 관리합니다. 이렇게 구조를 변경하게 되면 빈번한 조인 연산과 과도한 네트워크 트래픽 없이 메인 데이터베이스 내부에서 연산 처리가 가능하게 됩니다.



▼ 연습문제

☑️ 문제 1

Q. 문제	분산 데이터베이스의 특징으로 옳지 않은 것은?
A. (1)	논리적으로 동일한 시스템에 속한다
A. (2)	물리적으로 분산되어 있는 데이터들의 모임이다
A. (3)	데이터 베이스를 각각 여러곳의 가상 시스템으로 사용할 수 있도록 한다
A. (4)	데이터베이스를 하나의 가상 시스템으로 사용할 수 있도록 한다

▼ 정답

(3)

☑️ 문제 2

Q. 문제	분산 데이터베이스의 종류로 옳지 않은 것은?
A. (1)	분할 투명성
A. (2)	위치 투명성
A. (3)	지역 사상 투명성
A. (4)	비장애 투명성

▼ 정답

(4) 장애 투명성

☑️ 문제 3

Q. 문제	분산 데이터베이스의 장점으로 옳지 않은 것은?
A. (1)	지역 자치성, 점증적 시스템 용량 확장
A. (2)	빠른 응답 속도 & 통신 비용 절감
A. (3)	설계 및 관리의 편이성
A. (4)	시스템 규모의 적절한 조절

▼ 정답

(3) 분산 데이터베이스의 단점으로 설계 및 관리의 복잡성과 비용이 있다