2주차

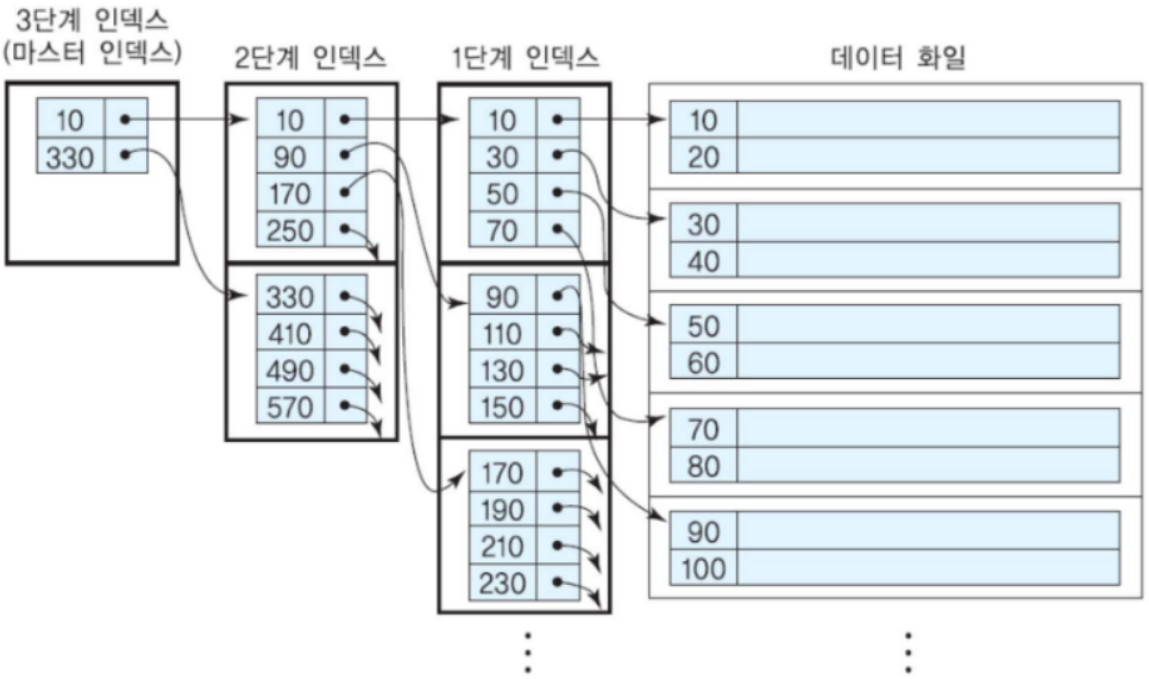
상태	시작 전
주제	데이터베이스

2주차(2024.06.28)

- 7 물리적 데이터베이스 설계 (2/2)
- 8 릴레이션 정규화

다단계 인덱스

- 인덱스 자체가 크면 인덱스 탐색 시간이 오래 걸림
- 인덱스를 탐색하는 시간을 줄이기 위해서 단일 단계 인덱스를 디스크 상의 하나의 순서 파일로 간주하고 단일 단계 인덱스에 대해 다시 인덱스를 정의할 수 있음
- 다단계 인덱스는 가장 상위 단계의 모든 인덱스 엔트리들이 한 블록에 들어 갈 수 있을때까지 이 과정 반복
- 가장 상위 인덱스는 마스터 인덱스
- 마스터 인덱스는 하나의 블록단위로 이루어져 주 메모리에 상주할 수 있음
- 다단계 인덱스는 B+ 트리를 사용함



- 인덱스는 키값에 sorting이 되어있음
- 2단계 인덱스는 희소 인덱스로 만들 수 있음
- 마지막 단계 인덱스는 한 블록에 다 들어가있음

💡 인덱스 maintenance overhead

- 인덱스를 사용하는 이유는 검색에 장점이 있기 때문
- 인덱스 삽입, 수정, 삭제 작업 시 성능이 저하됨
- 인덱스는 데이터 테이블 외에도 별도의 저장 공간을 차지하여 인덱스가 많아질수록 추가적 디스크 공간 필요
- 데이터베이스에서 쿼리 최적화를 위해 인덱스의 통계 정보를 유지하는데 이 통계는 주기적으로 업데이트 되어 오버헤드가 발생한다.
- 인덱스가 많으면 데이터베이스 동시성이 높은 환경에서는 인덱스 유지보수로 인해 잠금 경합이 발생할 수 있다. → 데드락 발생 가능성

이러한 오버헤드를 줄이기 위한 방법

- 무분별한 인덱스 생성이 아닌 필요한 인덱스만 생성한다.
- B-tree, Hash index, Bitmap index 등 다양한 인덱스 유형의 구조를 최적화한다.
- 인덱스 사용을 모니터링하고 주기적인 인덱스 튜닝을 통해 최적의 성능을 유지하도록 한다.
- 인덱스 재구성을 통해 통계 정보 업데이트 작업을 데이터베이스 사용량이 적은 시간대에 수행하도록 스케줄링한다.

장점

- 인덱스가 크면 인덱스를 탐색하는 시간도 오래 걸림
- 이 시간을 줄이기 휘해서

B+ 트리

SQL의 인덱스 정의

CREATE INDEX 인덱스이름 **ON** [속성(1개 혹은 여러개)];

```
CREATE INDEX EMPLOYEE ON (DNO, SALARY);
# 다수 속성을 사용한 인덱스 정의
```

- 다수 속성 인덱스 사용 쿼리 예시

```
SELECT *
FROM EMPLOYEE
WHERE DNO = 3 AND SALARY = 4000000;
# equal 사용
```

```
SELECT *
FROM EMPLOYEE
WHERE DNO >= 2 AND SALARY <= 4000000;
# range 쿼리 사용
# sorting이 되어있어 인덱스활용에 장점
```

```
SELECT *
FROM EMPLOYEE
WHERE DNO = 2 OR DNO = 3;
```

```
SELECT *
FROM EMPLOYEE
WHERE SALARY >= 2000000 AND SALARY <= 4000000;
# 사용 불가 -> SALARY로만은 사용 불가능
```

인덱스의 장점과 단점

- 장점 : 검색 속도를 향상, 소수의 레코드들을 수정하거나 삽입하는 갱신 작업의 연산 속도를 향상
- 단점 : 인덱스 파일 공간을 따로 쓰므로 주·보조 기억장치 공간을 더 씀

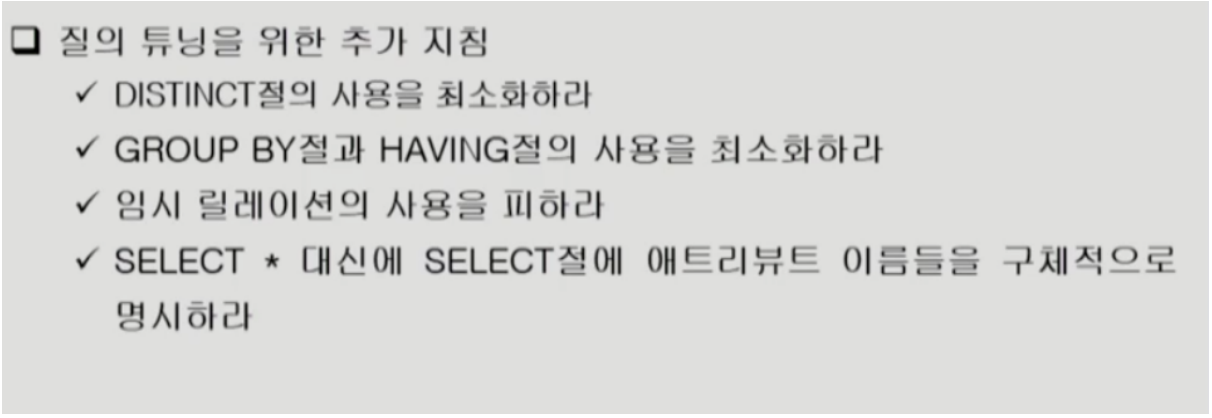
인덱스 선정 지침과 데이터베이스 튜닝

- 가장 중요한 쿼리들과 수행 빈도, 가장 중요한 갱신들과 이들의 수행빈도 등 질의와 갱신에 대한 바람직한 성능을 고려하여 인덱스를 선정한다.

- 워크로드 내의 각 질의에 대해 이 질의가 어떤 릴레이션들에 접근되는가, 어떤 애트리뷰트들을 검색하는가, WHERE절의 선택/조인 조건에 어떤 애트리뷰트가 포함되는가, 등등등 고려하기도 한다.
- 어떤 릴레이션에 인덱스를 생성해야 하는가, 어떤 애트리뷰트를 탐색키로 선정하는가, 몇 개의 인덱스를 생성하는가, 각 인덱스를 클러스터링,밀집/희소 인덱스 중 어느 유형을 택할 것인지도 고려

❑ 인덱스를 결정하는데 도움이 되는 몇 가지 지침
- ✓ 지침 1: 기본 키는 클러스터링 인덱스를 정의할 훌륭한 후보
- ✓ 지침 2: 외래 키도 인덱스를 정의할 중요한 후보
- ✓ 지침 3: 한 애트리뷰트에 들어 있는 상이한 값들의 개수가 거의 전체 레코드 수와 비슷하고, 그 애트리뷰트가 동등 조건에 사용된다면 비 클러스터링 인덱스를 생성하는 것이 좋음
- ✓ 지침 4: 투플이 많이 들어 있는 릴레이션에서 대부분의 질의가 검색하는 투플이 2% ~ 4% 미만인 경우에는 인덱스를 생성
- ✓ 지침 5: 자주 갱신되는 애트리뷰트에는 인덱스를 정의하지 않는 것이 좋음
- ✓ 지침 6: 갱신이 빈번하게 이루어지는 릴레이션에는 인덱스를 많이 만드는 것을 피함

❑ 인덱스를 결정하는데 도움이 되는 몇 가지 지침(계속)
- ✓ 지침 7: 후보 키는 기본 키가 갖는 모든 특성을 마찬가지로 갖기 때문에 인덱스를 생성할 후보
- ✓ 지침 8: 인덱스는 화일의 레코드들을 충분히 분할할 수 있어야 함
- ✓ 지침 9: 정수형 애트리뷰트에 인덱스를 생성
- ✓ 지침 10: VARCHAR 애트리뷰트에는 인덱스를 만들지 않음
- ✓ 지침 11: 작은 화일에는 인덱스를 만들 필요가 없음
- ✓ 지침 12: 대량의 데이터를 삽입할 때는 모든 인덱스를 제거하고, 데이터 삽입이 끝난 후에 인덱스들을 다시 생성하는 것이 좋음
- ✓ 지침 13: ORDER BY절, GROUP BY절에 자주 사용되는 애트리뷰트는 인덱스를 정의할 후보



💡 인덱스가 사용되지 않는 경우

- 시스템 카탈로그(메타 데이터)가 오래전의 데이터베이스 상태를 나타낼 경우 →
- DBMS 질의 최적화 모듈이 릴레이션의 크기가 작아서 인덱스가 도움이 되지 않는다고 판단
- 인덱스가 정의된 속성에 산술 연산자가 사용된 경우 ex) 속성 * 10 > 4000000
- DBMS가 제공하는 내장함수가 집단함수 대신에 사용된 경우 ex) SUBSTRING
- Null 값에 대해서는 인덱스가 사용되지 않는다.

B+ 트리 - 인덱스

Table에 저장되어 있는 Record 중에 원하는 Record를 빠르게 찾기 위해 사용 : OlogN

대부분 B-Tree와 같은 자료구조를 사용해서 구현되어 있음

인덱스는 B-Tree와 같은 자료구조로 정렬되어 있어 탐색이 빠르다.

B-Tree VS Hash Table

- Index를 Hash Table로 구현하면 탐색에 O(1)의 시간 복잡도로 탐색이 가능
- Hash는 Range Scan이 불가능 하고 Like 검색도 인덱스를 사용할 수 없다.
- 복합키 Index 사용에도 불리함

    - B-Tree 에서는 A, B 컬럼을 복합키로 Index를 생성했을 때, Where 절에 A컬럼 조건만 있더라도 Index를 사용해서 효율적으로 탐색이 가능
    - Hash는 A, B를 복합키로 Index를 생성하면 A, B 둘다 where 절에 동등조건으로 걸려 있어야만 사용이 가능
- 즉 Hash Table로 인덱스를 구현하면 단 하나의 데이터만 탐색할 때 효율적이고 B-Tree에 비해 득보다 실이 많음

B-Tree VS BST(RedBlack-Tree)

- BST는 Range Scan, Like 검색 가능, BST중에서도 가장 빠름
- 저장 매체에 따라 사용하는 자료구조가 달라진다.
- HDD(하드 디스크 드라이브)는 순차 읽기 성능은 준수하지만 Random Access는 상대적으로 느리다
- 이때 사용되는게 B-Tree인데 B-Tree가 RedBlack-Tree에 비해 Random Access가 적고 순차 읽기가 많음
- 하드 디스크는 읽기 성능에 비해 쓰기 성능이 느려서 CUD 발생 시 Random Access와 쓰기가 상대적으로 많이 발생하는 RedBlack-Tree보다는 B-Tree가 더 좋은 성능을 보여줌
- In-Memory가 아닌 디스크에 데이터가 저장되는 상황에서는 상대적으로 Random Access가 적고, CUD에 오버헤드가 적은 B-Tree가 BST보다 성능이 좋음

릴레이션 정규화

릴레이션 정규화

- 부주의한 데이터베이스 설계로 제어할 수 없는 데이터 중복을 야기하여 여러가지 **갱신 이상**을 유발한다.
- 데이터베이스 정규화를 통해 **주어진 릴레이션 스키마를 함수적 종속성과 기본 키를 기반으로 분석하여, 원래의 릴레이션으로 분해함으로써 중복과 세 가지 갱신이상을 최소화 하는것이다.**

- 좋은 데이터베이스란? 정보의 중복과 갱신 이상이 생기지 않고, 정보의 손실을 막으며, 실 세계를 훌륭히 표현했으며, 애트리뷰트들 간의 관계가 잘 표현되는것을 보장하며 어떤 무결성 제약 조건의 시행을 간단하게 하여 효율성까지 고려하는것을 말한다.

> 💡 갱신이상 (Update anomaly)
>
> - **수정이상 : 반복된 데이터 중에 일부만 수정하면 데이터 불일치가 발생**
> - **삽입이상 : 불필요한 정보를 함께 저장하지 않고는 어떤 정보를 저장하는것이 불가능**
> - **삭제이상 : 유용한 정보를 함께 삭제하지 않고는 어떤 정보를 삭제하는 것이 불가능**

사원

사원이름	사원번호	주소	전화번호	부서번호1	부서이름1
김창섭	2106	우이동	726-5869	1	영업
김창섭	2106	우이동	726-5869	2	기획
박영권	3426	사당동	842-4538	3	개발
이수민	3011	역삼동	579-4685	2	기획
이수민	3011	역삼동	579-4685	3	개발

- **수정 이상**: 박영권의 부서이름을 개발에서 운영으로 바꾸면 이수민의 부서번호3의 개발과 이름이 겹쳐서 갱신 이상 발생
- **삽입 이상**: 새로운 4 운영이라는 부서를 개설했는데 그 부서에 해당하는 사람이 없다면 갱신 이상 발생
- **삭제 이상**: 첫째줄 튜플을 삭제하면 1 영업부의 정보가 사라져 갱신 이상 발생

함수적 종속성

- 정규화 이론의 핵심

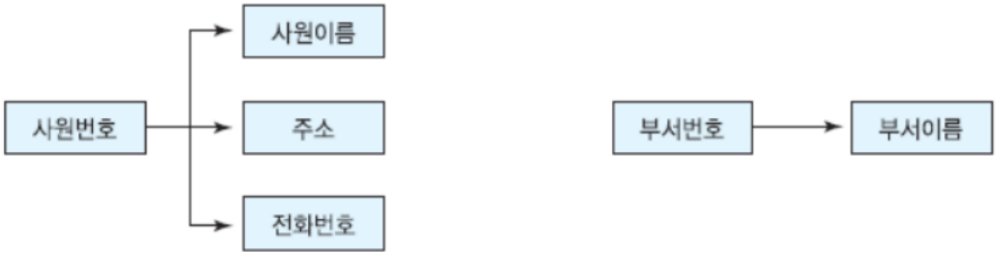
- 애트리뷰트 간의 관계를 나타냄
- 제2정규형부터 BCNF까지 적용됨

결정자

- 어떤 애트리뷰트의 값은 다른 애트리뷰트의 값을 고유하게 결정
- 아래 그림에서 사원번호는 사원이름을 고유하게 결정
- A가 B를 결정한다, A가 B의 결정자이다, A→B로 표현사원번호→사원이름, 주소, 전화번호, 부서이름부서번호→부서이름

사원

사원번호	사원이름	주소	전화번호	직책	부서번호	부서이름
4257	정미림	홍제동	731-3497	팀장	1	홍보
1324	이범수	양재동	653-7412	프로그래머	2	개발
1324	이범수	양재동	653-7412	웹 디자이너	1	홍보
3609	안명석	양재동	425-8520	팀장	3	홍보

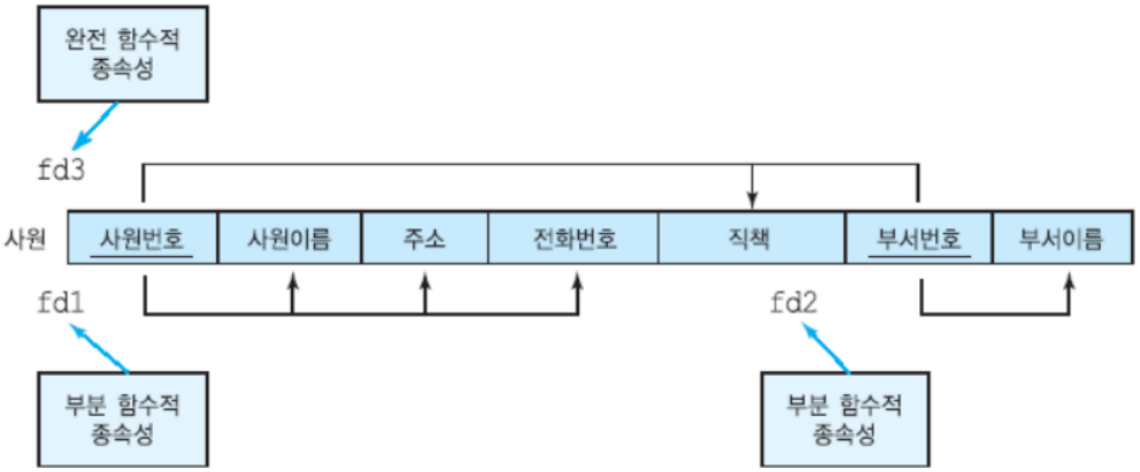


완전 함수적 종속성

- 기본키에 해당하지 않는 애트리뷰트가 기본키에 함수적 종속성을 가짐
- 직책은 기본키인 (사원번호, 부서번호)가 결정자
- 부서번호→직책

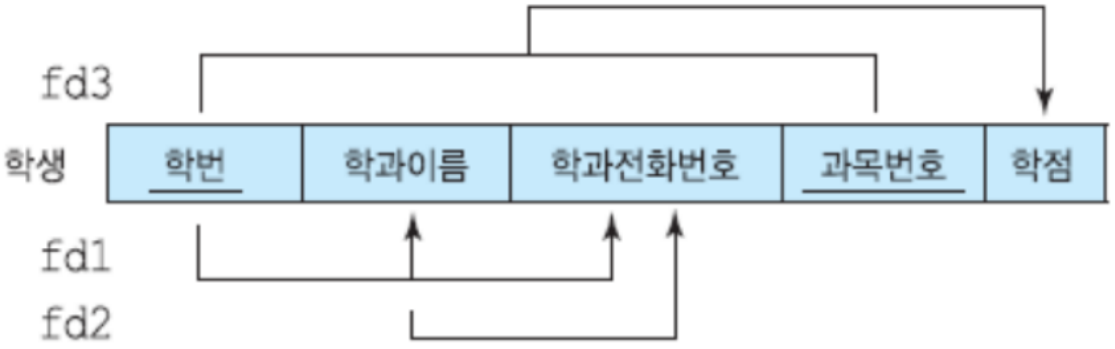
부분 함수적 종속성

- 기본키가 복합키일 때 부분키로만 다른 애트리뷰트를 결정할 수 있는 함수적 종속성
- 사원이름, 주소, 전화번호는 부분키인 사원번호가 결정자
- 부서이름은 부분키인 부서번호가 결정자



이행적 함수적 종속성

- A, B, C 애트리뷰트에서 A→B∪B→C와 A→C가 필요충분조건

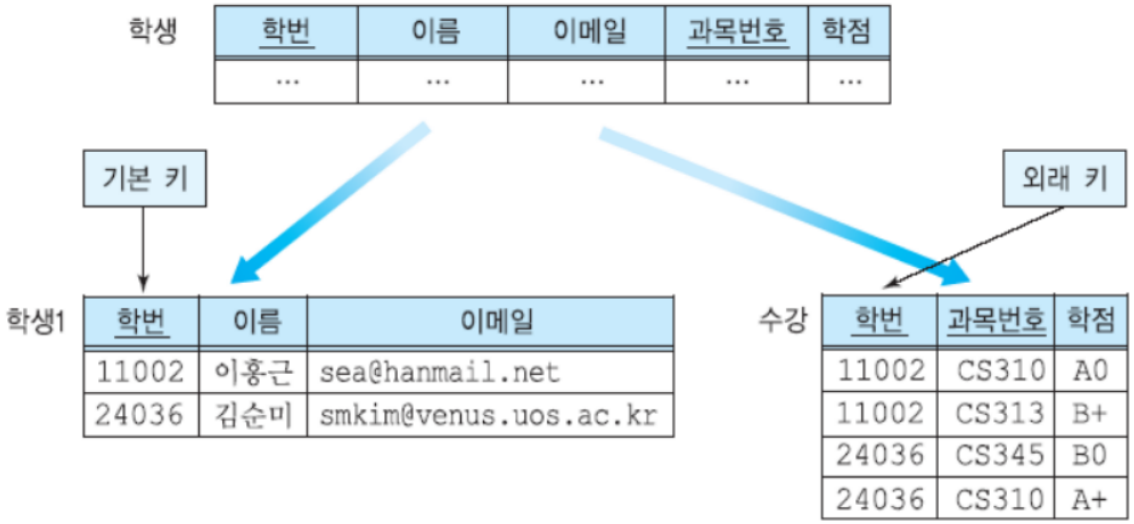


릴레이션 분해

- 한 개의 릴레이션을 여러 개의 릴레이션으로 분해
- 중복과 갱신 이상을 줄일 수 있는 장점
- 분해된 릴레이션으로 원래 릴레이션을 재구성하지 못하는 단점
- 원래는 조인이 필요없었던 질의가 분해 후에는 필요하게 되는 단점

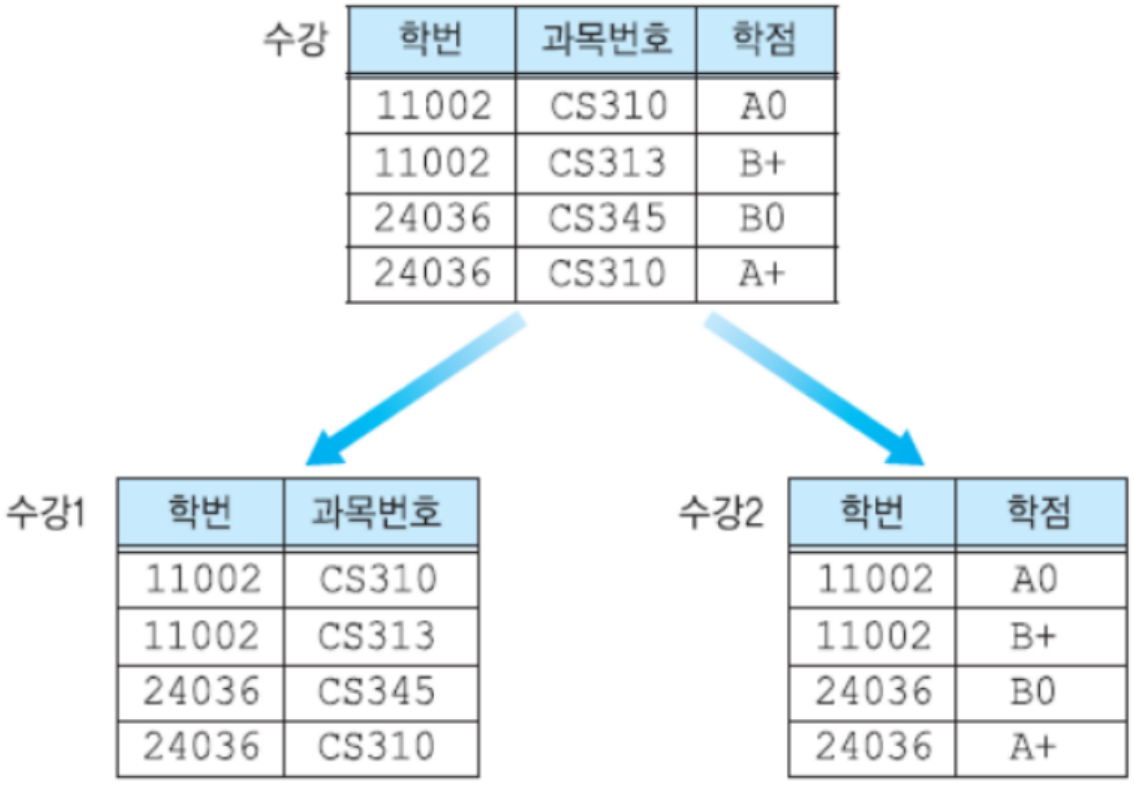
무손실 분해

- 분해된 릴레이션을 조인했을 때 원래 릴레이션이 나오는 분해
- 분해되었을 때 정보가 손실되지 않아서 무손실



가짜 튜플

위의 수강 릴레이션에서 한 번 더 분해 후 조인하면 가짜 튜플 발생



학번	과목번호	학점
11002	CS310	A0
11002	CS310	B+
11002	CS313	A0
11002	CS313	B+
24036	CS345	B0
24036	CS345	A+
24036	CS310	B0
24036	CS310	A+

정규형 종류

제 1정규형

릴레이션에서 다치 애트리뷰트를 단일 애트리뷰트로 바꾸는 정규형

학생

학번	이름	과목번호	주소
11002	이홍근	{CS310,CS313}	우이동
24036	김순미	{CS310,CS345}	양재동

위 그림은 다치 애트리뷰트가 존재하므로 제 1정규형이 아님

학생

학번	이름	과목번호	주소
11002	이홍근	CS310	우이동
11002	이홍근	CS313	우이동
24036	김순미	CS345	양재동
24036	김순미	CS310	양재동

과목번호 집합을 원소들로 분해함으로써 제 1정규형을 만족

제 2정규형

제 1정규형을 만족하고, 후보키가 아닌 애트리뷰트가 기본키에 완전 함수적 종속성을 가짐

학생1

학번	학과이름	학과전화번호
11002	컴퓨터과학	210-2261
24036	정보통신	210-2585
11048	컴퓨터과학	210-2261

학번→학과이름 / 학과이름→학과전화번호 이므로 학번→학과전화번호도 만족

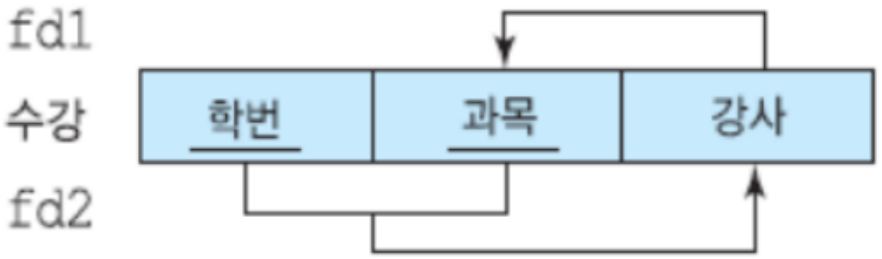
제 3정규형

제 2정규형을 만족하고, 이행적 함수적 종속성을 제거한 것

수강

학번	과목	강사
11002	데이터베이스	이영준
11002	운영 체제	고성현
24036	자료 구조	엄영지
24036	데이터베이스	조민형
11048	데이터베이스	이영준

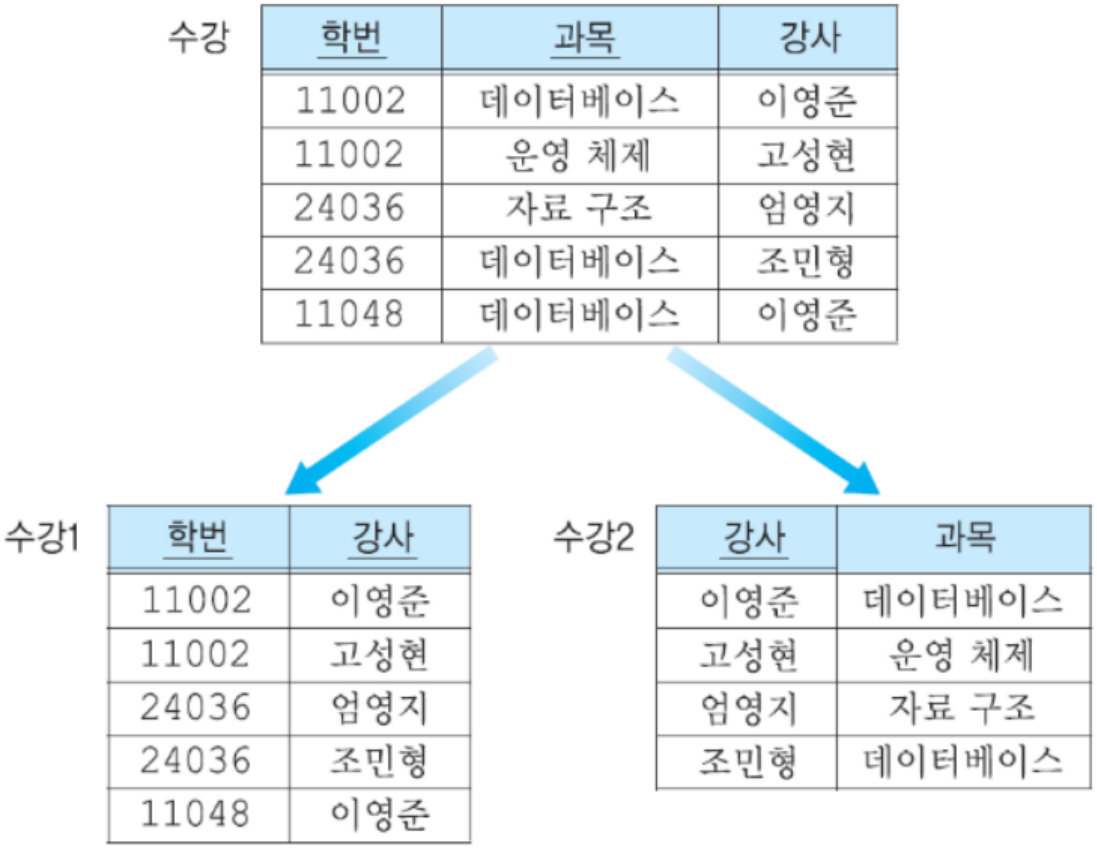
[그림 7.20] 제3정규형을 만족하는 릴레이션



[그림 7.21] 수강 릴레이션에 존재하는 함수적 종속성

BCNF

- 제 3정규형을 만족하고, 모든 결정자가 후보키인 것
- 대부분의 제 3정규형은 BCNF도 만족분해 전은 BCNF를 만족하지 않지만, 분해함으로써 만족



정규화 장점

- 중복과 갱신 이상 감소
- 정규화 단계가 올라갈 수록 무결성 제약조건을 위한 코드가 감소

정규화 단점

- 높은 정규형이 꼭 최적인 것은 아닐 때도 있어서 필요없는 과정일 수 있음
- 원래 릴레이션은 조인이 필요없었는데 분해함으로써 필요한 경우 발생