NHN Academy

Module 10
스키마 정제와 정규형

Module 10: 스키마 정제와 정규형

개념 설계 단계를 거치면 릴레이션 스키마의 집합과 무결성 제약 조건이 산출되어, 최종적인 데이터베이스 설계 내역을 만들어 낼 수 있게 됩니다. 초기 설계 내역은 무결성 제약조건들을 ER 모델에서 가능했던 것 보다 면밀하게 정제해 주어야 합니다. 이 단원에서는 개념적 스키마를 정제하는 법을 살펴봅니다.

이 장을 마치면, 다음과 같은 것들을 할 수 있게 됩니다:

- 데이터베이스 설계 단계에서 스키마 정제 단계를 이해하게 됩니다.
- 함수 종속에 대해 이해하고 설명할 수 있습니다.
- 정규형을 식별하고 위반을 검출할 수 있습니다.
- 각각의 정규형을 정의할 수 있습니다.
- 정규화를 수행하여 정규형을 만들어 낼 수 있습니다.
- 정규화 단계에 대해 설명할 수 있습니다.

목차

스키마 정제 개요

개념 설계에서 표현하지 못하는 문제들은 스키마 정제를 통해 해결할 수 있습니다. 정보를 중복해서 저장하는 것이 이러한 문제의 근본 원인이 되는데, 여기에서는 분해를 이용해 이 문제들과 그 에서 비롯되는 또 다른 문제를 해결하는 방법을 학습합니다.

Table of Contents

스키마 정제 개요

- 정제되지 않은 스키마에서의 문제
  - 중복 이상
    어떤 데이터는 반복적으로 저장됨
  - 갱신 이상
    반복 저장된 데이터 중 한 투플을 갱신할 때 다른 모든 사본을 갱신하지 않으면 불일치 발생
  - 삽입 이상
    한 정보를 저장하려면 다른 정보도 같이 저장하여야 함
  - 삭제 이상
    어떤 정보를 지우면 다른 정보도 같이 삭제됨

스키마 정제가 필요하게 되는 문제들은 정보를 중복해서 저장하는데서 발생합니다. 분해를 하면 중복성은 없앨 수 있지만 그 자체의 문제점을 야기할 수도 있습니다.

중복성으로 발생하는 문제

- 중복 저장(Duplicate)
  어떤 정보는 반복적으로 저장된다.
- 갱신 이상(Update Anomaly)
  반복 저장된 데이터 중 한 사본을 갱신할 때 다른 모든 사본도 갱신하지 않으면 불일치가 발생한다.
- 삽입 이상(Insert Anomaly)
  어떤 정보를 저장하려면 다른 정보도 같이 저장해야 한다.
- 삭제 이상(Delete Anomaly)
  어떤 정보를 지우면 다른 정보도 같이 삭제된다.

아래와 같은 개체 집합의 경우를 생각해봅시다.

`Parttime_emp(EmpID, Name, Parkingslot, Grade, WagePerHr, Workingtime)`

Partime_emp의 키는 EmpID입니다. WagePerHr 속성이 Grade 속성에 의해 결정된다고 가정합시다. 즉 WagePerHr는 Grade에 의해 결정되고 한 Grade에 해당하는 WagePerHr는 하나밖에 존재하지 않는 경우입니다. 이러한 무결성 제약조건이 바로 함수 종속성(Functional Dependency)의 한 예입니다. 이에 따라 Parttime_emp 릴레이션에는 중복성이 생길 수 있습니다.

EmpID	Name	Parkingslot	Grade	WagePerHr	Workingtime
1	홍길동	16	8	10,000	40
2	이순신	13	8	10,000	30
3	김영랑	87	5	7,000	30

4	박목월	37	5	7,000	20
5	천상병	25	8	10,000	40

만일 두 투플의 등급 필드에 같은 값이 저장된다면 이 무결성 제약 조건에 따라 WagePerHr 필드에도 같은 값이 저장되어야 합니다. 이러한 중복성 때문에 다름과 같은 결과가 나타납니다.

- 어떤 정보는 여러 번 저장될 수 있습니다. Grade 값 8은 WagePerHr 10,000과 대응하는데 이 연관성이 세 번 반복됩니다. 같은 정보를 여러 번 저장함에 따른 공간의 낭비도 문제지만, 중복 저장하게 되면 잠재적으로 불일치가 발생할 수 있습니다.
- 첫 번째 투플의 WagePerHr 값은 갱신하였으나 두 번째 투플의 값은 갱신하지 않았다면 갱신 이상이 발생합니다.
- 한 직원의 투플을 삽입하려면 직원의 Grade 값에 따른 WagePerHr 값을 알아야 합니다. 삽입 이상이 발생합니다.
- 어떤 등급 값에 대한 모든 투플을 삭제한다면 그 등급 값과 그에 따른 WagePerHr의 연관성을 잃어버리게 됩니다. 삭제 이상이 발생합니다.

분해법

- 속성을 부자연스럽게 묶어서 한 릴레이션 스키마로 만들면 중복성이 발생
- 함수 종속(Functional Dependency)를 이용하여 상황을 식별
  - 릴레이션을 더 작은 릴레이션의 모임으로 대치
  - 작은 일레이션은 본래 릴레이션 속성의 부분집합으로 이루어 짐
- 분해의 문제
  - 릴레이션의 분해가 필요한가?
    - 제안된 정규형(Normal Form)으로 분해
  - 분해했을 때 발생할 수 있는 문제는 무엇인가?
    - 무손실 조인(Lossless Join) 성질에 따라 릴레이션 인스턴스 복구
    - 종속성 유지(Dependency Preservation) 조건에 따라 계약 조건 유지

속성들을 부자연스럽게 묶어서 한 릴레이션 스키마로 만들면 중복성이 발생합니다. 함수 종속 관계를 이용하면 이러한 상황을 식별해 낼 수 있고 스키마를 정제할 필요성이 있는지를 파악할 수 있습니다. 기본적으로, 릴레이션을 더 작은 릴레이션으로 대치하면 중복성으로 비롯되는 많은 문제들이 처리될 수 있다는 것입니다. 더 작은 릴레이션은 원본 릴레이션의 진부분집합으로 이루어집니다. 이 과정을 큰 릴레이션을 여러 작은 릴레이션으로 분해(Decomposition)한다고 말합니다.

Parttime_emp 릴레이션을 아래와 같이 두 릴레이션을 분해하여 중복성을 피할 수 있습니다

```
Parttime_emp(EmpID, Name, Parkingslot, Grade, WagePerHr, Workingtime)
HourlyWage(Grade, WagePerHr)
```

Parttime_emp 인스턴스에 해당하는 이들 릴레이션의 인스턴스는 아래와 같습니다.

EmpID	Name	Parkingslot	Grade	Workingtime
1	홍길동	16	8	40
2	이순신	13	8	30
3	김영랑	87	5	30
4	박목월	37	5	20
5	천상병	25	8	40

Grade	WagePerHr
8	10,000
5	7,000

HourlyWage 릴레이션에 투플 하나를 추가하면 어떠한 등급에 해당하는 시간당 임금도 쉽게 기록해 둘 수 있으며, Parttime_emp 릴레이션에 그 등급에 해당하는 직원이 없어도 무방합니다. Grade 값에 해당하

는 직원이 없어도 무방합니다. Grade에 해당되는 WagePerHr 값을 변경하면 해당 임금이 변경됩니다. 이는 여러 투플을 갱신하는 것 보다 훨씬 효율적이고 불일치의 위험도 줄어듭니다. 삽입 이상과 삭제 이상 또한 없어집니다.

분해에 따르는 문제

어떤 릴레이션의 스키마를 분해할 때 주의하지 않으면 원래보다 더 많은 문제가 발생할 수 있습니다.

- 릴레이션의 분해가 필요한가?
- 릴레이션을 분해했을 때 발생할 문제는 무엇인가?

릴레이션의 분해가 필요한지에 대해서는, 릴레이션에 대한 몇 가지 정규형(Normal Form)들이 제안되어 있습니다. 어떤 릴레이션 스키마가 어느 한 정규형에 속한다면 특정 종류의 문제는 발생하지 않는다는 것을 보장할 수 있습니다. 주어진 릴레이션 스키마가 어떤 정규형에 속하는지 알고 있다면 더 분해할 필요가 있는지를 결정하는데 도움이 됩니다.

릴레이션을 분해했을 때 발생할 문제가 있는지에 관해서는, 무손실 조인(Lossless-Join) 성질에 따라 분해된 릴레이션은 항상 복구할 수 있고, 종속성 유지(Dependency-Preservation) 성질에 따라 분해된 릴레이션 각각에 대해 일정 제약조건을 걸어주면 원래 릴레이션의 모든 제약조건이 그대로 유지될 수 있습니다.

분해의 큰 결점중의 하나는 분해된 릴레이션을 조인해야 할 때가 많다는 것입니다. 이런 조인이 빈번하게 발생한다면 릴레이션을 분해함으로써 생기는 성능상의 손실이 있을 수 있습니다. 이런 경우 중복성의 문제를 놓아두고 릴레이션을 분해하지 않기로 결정하는 경우도 있습니다. 설계에 내재된 중복성으로 인해 야기될 수 있는 잠재적인 문제들을 충분히 인지하고 이를 피하기 위한 절차를 수행하는 것이 중요합니다.

데이터베이스 설계를 위해서는 정규형에 대한 정확한 이해가 필요합니다. 정규형으로 방지할 수 있는 문제가 무엇인지, 또는 할 수 없는 문제가 무엇인지 정확히 알고 있어야 하며, 분해의 기법과 분해로 생길 수 있는 잠재적인 문제들이 무엇인지 알고 있어야 합니다.

함수 종속

- 함수 종속(Functional Dependency)는 일종의 제약 조건
- 어떤 릴레이션 스키마를 R이라고 하고 X와 Y를 R에 속한 속성 집합이라고 하고, R의 인스턴스 r에 속한 투플 t1과 t2가 다음의 조건을 만족하면 FD X → Y가 존재한다고 한다

  t1.X = t2.X 이면 t1.Y = t2.Y

- 한 릴레이션에 대해 적법한 인스턴스가 되려면 명세된 모든 FD를 만족해야 함
- 기본 키는 FD의 특수한 경우
  - 키에 대한 속성은 X의 역할을 하게 되며, 나머지 속성은 Y의 역할을 함
  - X → Y가 만족되고 X의 진부분집합 V가 있어서 V → Y가 만족되면 X는 수퍼키이지 키는 아님

함수 종속(Functional Dependency)는 일종의 무결성 제약조건으로, 키의 개념을 일반화한 것입니다.

어떤 릴레이션 스키마를 R이라 하고 X와 Y를 R에 속한 속성의 집합인데 공집합이 아닌 경우, R의 한 인스턴스 r에 속한 어떤 투플의 쌍 t1과 t2가 다음 조건을 만족하면 FD X → Y가 존재한다고 합니다.

`t1.X = t2.Y` 이면 `t1.Y = t2.Y` 이다.

여기에서 t1.X의 의미는 투플 t1을 X에 속한 속성들로 프로젝션 한다는 것으로서, 투플 관계 해석의 표기법 t.a가 투플 t의 속성 a를 지칭한다는 것을 확장한 것입니다. FD X → Y는 두 투플의 속성들의 값이 같다면 Y 속성의 값도 같아야 한다는 것을 의미합니다.

아래 릴레이션은 AB → C라는 함수 종속을 만족합니다.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a4	b1	c3	d1

상위 두 투플은 FD가 키 제약조건과 같은 것이 아니라는 것을 보여줍니다. FD를 위반하지 않으면서도 AB는 이 릴레이션의 키가 아닙니다. 세 번째와 네 번째 투플을 보면 A 필드와 B 필드 중 하나라도 값이 달라지면 C 필드의 값도 달라질 수 있음을 알 수 있습니다. 투플 <a1, b1, c3, d1>을 추가하면 이 FD를 위반합니다.

어떤 릴레이션에 대해 적법한 인스턴스가 되려면 명세된 모든 FD를 비롯한 모든 무결성 제약조건을 만족해야 합니다. 몇 개의 릴레이션 인스턴스를 보고 그 FD를 만족한다고 바로 추론할 수는 없는데, FD는

무결성 제약 조건들과 마찬가지로 한 릴레이션의 모든 적법한 인스턴스에 대한 주문이기 때문입니다.

기본 키 제약조건은 FD의 특수한 경우입니다. 키에 속한 속성들은 X의 역할을 하게 되며, 그 릴레이션의 나머지 속성들은 Y의 역할을 하게 됩니다. 그렇지만 FD의 정의에서는 집합 X가 최소일 필요는 없는데 반하여, X가 키가 되려면 그 조건도 만족해야 합니다. 만일 $X \rightarrow Y$가 만족되고, Y가 모든 속성의 집합이며, X의 진부분집합 V가 있어서 $V \rightarrow Y$가 만족된다면 이때 X는 슈퍼 키이지, 키는 아닙니다.

스키마 정제

- 개체 집합 스키마 정제

  {EmpID} → {EmpID, Name, Parkingslot, Grade, WagePerHr, Workingtime}
  E E N P G W H

- 등급에 따라 시간당 임금이 결정될 경우 두 FD가 존재

  E → ENGWH G → W

- 부분적 함수 종속
  - 기본 키 구성 속성의 일부에 종속되거나, 기본 키가 아닌 다른 속성에 종속되는 경우
- 이행적 함수 종속
  - 함수 종속에 이행이 있을 때 (A → B, B → C = A → C)
- 완전 함수 종속
  - 함수 종속 X → Y에서 X로부터 속성 A를 제거하면 함수 종속 X → Y가 성립하지 않는 경우
  - 즉, 임의의 속성 A ∈ X 에 대해서 Y가 (X{A})에 함수 종속되지 않는 경우

ER 설계로 만들어내는 스키마에는 중복성 문제가 잠복해 있을 가능성이 있습니다. ER 설계는 복잡하고 주관적인 과정이며 또 어떤 제약조건들은 ER 다이어그램으로 표현할 수 없기 때문입니다.

개체 집합에 대한 제약조건

Parttime_emp 릴레이션을 다시 생각해보면, 속성 EmpID가 가지는 제약 조건은 다음과 같은 함수 종속으로 표현할 수 있습니다.

{EmpID} → {EmpID, Name, Parkingslot, Grade, WagePerHr, Workingtime}

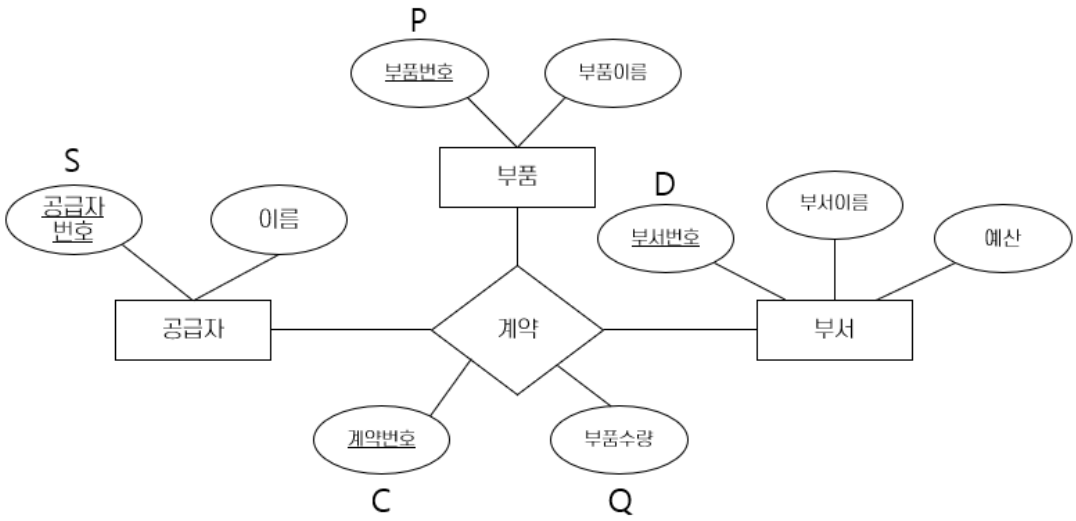
이 함수종속을 속성의 첫 문자만을 따서 E → ENPGWT로 표기할 수 있습니다. (양쪽에 속성 집합이 나올 수 있습니다) Parttime_emp 릴레이션에는 E → ENPGWT 함수 종속이 있고, 또 시간당 임금을 표시하는 WagePerHr 속성은 Grade 속성에 좌우되므로, G → W 함수 종속성이 생깁니다.

앞서 예에서 이 함수 종속 때문에 등급과 임금 관련하여 중복된 데이터를 저장하게 되었습니다. ER 모델에서는 키 제약조건이라는 특수한 경우만 지원하므로 이런 내용은 표시할 수 없습니다. ER 모델링 기간에는 Parttime_Emp라는 개체 집합에서는 이런 사실을 알아낼 수 없습니다.

이 문제는 애초의 설계가 잘못 되었기 때문이라고 생각할 수 있습니다. Wage 라는 개체 집합을 도입하고 Wage와 Parttime_Emp 사이에 관계를 넣으면 문제를 해결할 수 있다고 생각할 수도 있습니다. ER 모델링의 성질상 원래의 설계대로 가기가 쉽고, 따라서 이런 설계상의 문제를 찾아내고 더 좋은 설계를할 할 수 있도록 해주는 정규화된 기법을 사용할 수 있으면 큰 도움이 됩니다.

관계 집합에 대한 제약조건

아래와 같이 부품, 공급자, 부서 세 개체집합이 있고, 이 세 개체집합이 모두 관여하는 관계집합 계약이 있는 경우를 생각해봅시다.



계약 관계집합의 스키마를 CQPSD로 표현할 수 있습니다. 계약번호가 C인 계약은 공급자 S가 Q 만큼의 부품 P를 부서 D에 공급하기로 했음을 나타냅니다. 아래와

내부 방침상 한 부서가 한 공급자로부터 한 종류의 부품밖에 구매할 수 없다고 합시다. 그렇게 되면 동일한 공급자와 부서간에 여러 건의 계약이 있을 경우 이 모든 계약에서 부품은 항상 동일합니다. 이 제약조건은 FD가 되며, DS → P로 표시됩니다.

따라서 중복성이 발생하며 문제를 일으키게 됩니다. 이런 상황을 해결하려면 계약 릴레이션을 두 릴레이션으로 분해하여 하나는 CPSD, 하나는 SDP가 되게 해야 합니다. 릴레이션 SDP는 공급자가 부서에게 공급하는 부품을 기록하며 릴레이션 CPSD는 각 계약에 대한 추가 정보들을 기록합니다. ER 모델링 만으로는 이렇게 설계하기 어렵습니다.

함수 종속 이해

- 암스트롱의 공리
  - 반사(Reflexivity): X ⊇ Y 이면 X → Y
  - 첨가(Augmentation): X → Y 이면 어떠한 Z에 대해서도 XZ → YZ
  - 이행(Transitivity): X → Y이고 X → Z 이면 X → Z
- 이외의 규칙
  - 결합(Union): X → Y이고 X → Z 이면 X → YZ
  - 분해(Decomposition): X → YZ이면, X → Y 이고 X → Z

릴레이션 스키마 R에 대해 어떤 함수 종속의 집합이 주어진다면, 일반적으로 R에 대해 만족하는 다른 함수 종속들이 존재하게 됩니다. 아래와 같은 릴레이션 스키마가 있을 때,

```
Employee(EmpID, Name, ParkingSlot, DepartmentID, JoinDate)
```

각 부서는 하나의 공유하는 주차공간(ParkingSlot)을 가진다고 할 때, 다음과 같은 함수 종속을 생각할 수 있습니다. EmpID가 키이므로 EmpID → DepartmentID가 만족되며, FD DepartmentID → ParkingSlot도 만족되어야 합니다. 직원에 대한 적법한 인스턴스 내에서는 EmpID의 값이 동일한 투플들은 DepartmentID의 값도 동일해야 하며, 동일한 DepartmentID를 같는다면 동일한 ParkingSlot의 값도 동일해야 합니다. 따라서 EmpID → ParkingSlot도 만족됩니다.

f를 하나의 FD라고 하고, F를 어떤 FD의 집합이라고 합시다. 만일 F에 속한 모든 종속을 만족하는 어떤 릴레이션 인스턴스에서도 f가 만족한다면, 즉 F에 속하는 모든 함수 종속 관계가 만족할 때에는 언제나 f도 만족한다면 f는 FD의 집합 F에 의해 **암시된다(implied)**라고 합니다.

FD 집합에 대한 폐쇄

FD의 집합 F에 의해 암시되는 모든 FD의 집합을 F의 폐쇄(Closure)라고 하며 F^+로 표시합니다. FD의 집합 F에 대해 폐쇄를 구하는 작업을 추론(Inference)한다고 합니다. **암스트롱의 공리(Armstring's Axioms)**라고 불리는 세 법칙을 반복해서 적용하면 FD의 집합 F에 의해 암시되는 모든 FD들을 추론할 수 있습니다.

X, Y, Z는 릴레이션 스키마 R에 존재하는 속성의 집합들이라고 할 때,

- **반사(Reflexivity)**: X ⊇ Y 이면 X → Y
- **첨가(Augmentation)**: X → Y 이면 어떠한 Z에 대해서도 XZ → YZ
- **이행(Transitivity)**: X → Y이고 X → Z 이면 X → Z

FD의 한 집합 F가 적용될 때에는 F^+에 속하는 FD들만 생성되기 때문에 암스트롱의 공리들은 정당(Sound)합니다. 또한 이 세 가지 법칙을 반복해서 적용하면 폐쇄 F^+에 속하는 모든 FD들을 만들어 낼 수 있으므로 암스트롱의 공리들은 완전(Complete)합니다. 아래와 같은 규칙들도 알아두는 것이 좋습니다.

- **결합(Union)**: X → Y이고 X → Z 이면 X → YZ
- **분해(Decomposition)**: X → YZ이면, X → Y 이고 X → Z

이 두 법칙은 꼭 필요한 것은 아닌데, 암스트롱의 공리들을 사용해서 이들의 정당성을 증명할 수 있습니다.

이런 FD 추론 법칙들을 살펴보기 위해 어떤 릴레이션 스키마 ABC가 A → B이고 B → C인 FD들을 가지고 있다고 생각해봅시다. 화살표 오른쪽에 나타나는 속성들은 모두 왼쪽에도 있는 FD를 평범한 FD(Trivial FD)라고 하는데, 이런 종속 관계는 항상 만족됩니다. 반사 법칙을 사용하면 평범한 종속 관계를 모두 만들어 낼 수 있는데, 그 형태는 다음과 같습니다.

X → Y, 이때 Y ⊆ X, X ⊆ ABC, Y ⊆ ABC이다.

이행 법칙에 따라 A → C를 얻을 수 있습니다. 첨가 법칙을 사용하면 다음과 같은 비평범 종속 관계를 얻게 됩니다.

AC → BC, AB → AC, AB → CB

다음과 같이 구체화된 Contract 릴레이션을 살펴봅시다.

Contract(ContractNo, SupplierID, SubjectID, DepartmentID, PartNo, Quantity, ValuePrice)

Contract 릴레이션의 스키마를 CSJDPQV로 나타냅시다. 이 릴레이션에 속하는 한 투플의 의미는 공급자 S(SupplierID)가 부품 P(PartNo)를 Q(Quantity)개 만큼 부서(DepartmentID)가 수행하는 과제 J(SujectID)에 공급하기로 계약번호 C(ContractID)의 계약을 체결했음을 의미하고 이 계약의 금액 V는 ValuePrice 속성입니다.

다음의 무결성 제약조건이 만족된다는 것은 이미 알고 있습니다.

1. ContractID C는 키입니다. 따라서 C → CSJDPQV
2. 어떤 과제에서 어떤 부품을 구매하는 것은 하나의 계약으로 이루어집니다. JP → C
3. 각 부서는 각 공급자로부터 하나의 물품만을 구매할 수 있습니다. SD → P

이렇게 주어진 FD들의 집합에 대한 폐쇄에는 다음과 같은 FD들가 추론됩니다.

JP → C, C → CSJDPQV, 이행 법칙으로부터 JP → CSJDPQV가 추론됩니다.

SD → P, 첨가 법칙으로부터 SDJ → JP가 추론됩니다.

SDJ → JP, JP → CSJDPQV, 이행 법칙으로부터 SDK → CSJDPQV가 추론된다.

첨가 법칙이나 분해 법칙을 사용하면 이 폐쇄 안에 들어가는 FD들을 더 만들어 낼 수 있습니다. C → CSJDPQV에 분해 법칙을 적용하면 다음을 추론할 수 있습니다.

C → C, C → S, C → K, C → D 등

정규화와 정규형

어떤 릴레이션 스키마가 주어지면, 스키마의 설계가 올바른지 아니면 여러 개의 더 작은 릴레이션들로 분해하여야 하는지 판단하여야 합니다. 이에 대한 판단 기준으로 정규형들이 제안되었으며, 해당 릴레이션이 어떤 정규형에 속하는지에 따라 어떤 문제를 일으킬 수 있는지 알 수 있습니다. 이 단원에서는 정규형으로 릴레이션을 분해하는 과정인 정규화에 대해 알아봅니다.

Table of Contents

정규화(Normalization)

- 속성간의 종속성으로 인한 이상 현상이 발생하는 릴레이션을 분해하여 이상 현상을 없애는 과정
- 데이터의 중복 방지, 무결성을 충족하기 위한 데이터의 설계 방법
- 릴레이션 스키마가 어떤 정규형을 만족하는지 확인하는 테스트
- 정규화 원칙
  - 무손실 법칙: 분해된 릴레이션이 표현하는 정보는 분해되기 전의 정보를 모두 포함
  - 최소 데이터 중복 법칙: 이상 현상을 제거, 데이터 중복을 최소화
  - 분리 법칙: 독립된 함수 종속은 독립된 릴레이션으로 분해
- 장점
  - 이상 현상 해결
  - 새 속성 추가시 데이터베이스 변경의 최소화
  - 현실 세계의 개념간의 관계 표현

정규화(Normalization)는 릴레이션에 존재하는 속성간의 종속성으로 인한 이상 현상이 발생할 수 있는 경우를 방지하기 위해 릴레이션을 분해하여 이상현상을 없애는 과정입니다.

Dr. Codd가 처음 제안한 정규화 과정은 어떤 릴레이션 스키마가 어떤 정규형을 만족하는지 확인하는 일련의 테스트입니다. Codd는 제 1정규형, 제 2정규형, 제 3정규형이라고 이름 붙인 세 개의 정규형을 제안하였습니다. 그 후 Boyce와 Codd가 3정규형보다 더 강한 정의를 제안하고, BCNF(Boyce-Codd Normal Form)이라고 불렀습니다.

이 정규형들은 모두 한 릴레이션의 속성 사이의 함수 종속을 기반으로 합니다. 이와는 달리 4 정규형과 4 정규형은 각각 조인 종속성과 다치 종속성을 기반으로 합니다.

데이터의 정규화는 중복을 최소화하고, 삽입, 삭제, 갱신 이상을 최소화하기 위해 함수 종속과 기본 키를 기반으로 주어진 릴레이션 스키마를 분석하는 과정입니다. 어떤 조건(정규형 검사)들을 만족하지 못하는 적법하지 못한 릴레이션 스키마는 정규형 검사를 만족하는 더 작은 릴레이션 스키마들로 분해되어 적법한 특성을 갖게 됩니다. 정규화 과정은 다음 기능을 제공합니다.

- 릴레이션 스키마의 속성들 사이에 존재하는 함수 종속과 키들을 기반으로 그 스키마를 분석할 수 있는 정형적 구조
- 관계 데이터베이스를 임의의 수준으로 정규화할 수 있도록 개개의 릴레이션 스키마에서 수행할 수 있는 일련의 정규형 검사

한 릴레이션의 정규형은 그 릴레이션이 만족하는 가장 높은 정규형을 나타냅니다. 즉 그 릴레이션이 어느 정도까지 정규화 되었는가를 나타냅니다. 정규형들이 다른 요인들과 동떨어져서 고려되면 데잍베이스

설계를 보장하지 못합니다. 일반적으로 데이터베이스의 각 릴레이션 스키마가 BCNF 정규형인지 3 정규형인지 검사하는 것은 충분하지 않습니다. 그보다 분해에 의한 정규화 과정은 릴레이션 스키마들이 함께 가져야 하는 추가적인 특성들을 만족하도록 해야 합니다.

- 무손실 법칙: 분해된 릴레이션이 표현하는 정보는 분해되기 전의 정보를 모두 포함해야 한다.
- 최소 데이터 중복법칙: 이상 현상을 제거하여 데이터의 중복을 최소화한다.
- 분리 법칙: 독립된 함수 종속은 독립된 릴레이션으로 분해한다.

추가적인 형태의 제약조건을 기반으로 적법한 기준을 만족하기 위해 추가적인 정규형이 제안될 수 있습니다. 그러나 정규형은 제약조건들에 근거하여 정의되므로 데이터베이스 설계자와 사용자가 그 제약조건들을 이해하거나 발견하기 어렵다면 정규형의 실질적인 유용성은 떨어지게 됩니다. 현재는 BCNF나 3NF까지의 정규형만 고려하는 것이 일반적입니다.

다른 알아야 할 사항은 릴레이션을 항상 최상의 정규형으로 정규화 할 필요는 없다는 것입니다. 성능과 편의성을 위해서 릴레이션을 낮은 수준의 정규형으로 둘 수도 있습니다. 더 높은 정규형 릴레이션들을 조인한 결과를 기본 릴레이션으로 저장하는 과정을 비정규화(Denormalization)이라고 합니다.

제 1정규형(First Normal Form)

- 도메인의 원소들이 나눌 수 없는 단위로 되어 있을 때 그 도메인은 원자적(Atomic)이라고 함
- 어떤 릴레이션 R에속한 모든 도메인이 원자적일 때 R은 제 1정규형(1NF)에 속함

제 1정규형은 다른 정규형과는 달리 릴레이션에 대해 기본적인 요건만을 요구하며, 함수 종속 등의 다른 부가적인 정보를 요구하지 않습니다.

도메인의 원소들이 나눌 수 없는 단위로 되어 있을 때, 그 도메인은 원자적(Atomic)이라고 합니다. 릴레이션 스키마 R의 모든 원소들의 도메인이 원자적일 때 R이 제 1정규형(1NF)에 속한다고 합니다.

아래 릴레이션과 같이 한 도메인이 여러 값을 포함하고 있을 때, 이 집합은 원자적이지 않습니다.

학번	지도교수	학과	과목번호	성적
100	이순신	컴퓨터공학과	C102, D103	A, B
200	홍길동	컴퓨터공학과	C102	B
300	윤동주	기계공학과	D102	A
400	김영랑	수학과	F201	C

아래와 같이 릴레이션에 속한 모든 도메인이 원자적이면, 이 릴레이션은 제 1정규형에 속합니다.

학번	지도교수	학과	과목번호	성적
100	이순신	컴퓨터공학과	C102	A
100	이순신	컴퓨터공학과	D103	B
200	홍길동	컴퓨터공학과	C102	B
300	윤동주	기계공학과	D102	A
400	김영랑	수학과	F201	C

제 2정규형(Second Normal Form)

- 완전 함수 종속 개념에 기반을 둠
- 한 릴레이션이 1NF이고 기본 키에 속하지 않은 속성이 기본 키에 완전 함수 종속되는 경우

제 2정규형은 완전 함수 종속 개념에 기반을 둡니다. 완전 함수 종속(Full functional dependency)의 정의는 아래와 같습니다.

함수 종속 X → Y에서,

- X로부터 임의의 속성 A를 제거하면 X → Y가 성립되지 않는 경우
- 즉, 임의의 속성 A ∈ X에 대해서 Y가 (X {A})에 함수 종속하지 않는 경우

제 2 정규형 검사는 함수 종속의 왼편의 속성들이 기본 키의 일부인지 검사하는 것을 수반합니다. 만약 기본키가 단일 속성이라면 검사를 건너뛰어도 무방합니다. 릴레이션 스키마 R의 모든 속성이 R의 기본키에 대해서 완전 함수 종속이면 R을 제 2정규형(Second Normal Form – 2NF)이라고 합니다.

아래와 같은 릴레이션 스키마 Grade가 있고,

```
Grade(StudentNo, Professor, Department, SubjectID, Grade)
        S           P          D           J       G
```

릴레이션 인스턴스가 아래와 같다고 할 때,

StudentNo	Professor	Department	SubjectID	Grade
100	이순신	컴퓨터공학과	C102	A
100	이순신	컴퓨터공학과	C103	B
200	홍길동	컴퓨터공학과	C102	B
300	윤동주	기계공학과	D102	A
400	김영랑	수학과	F201	C

릴레이션 스키마 Grade는 아래와 같은 함수 종속을 가집니다.

FD1: S → PD

FD2: J → G

릴레이션 스키마가 제 2 정규형이 아니라면 여러 개의 제 2정규형 릴레이션으로 정규화 할 수 있습니다. 속성이 그들이 완전 함수 종속관계가 있는 기본 키의 일부 속성들과만 연관되도록 구성됩니다.

```
Student(StudentNo, Professor, Department)
Grade(StudentNo, Grade)
```

StudentNo	Professor	Department
100	이순신	컴퓨터공학과
200	홍길동	컴퓨터공학과

300	윤동주	기계공학과
400	김영랑	수학과

StudentNo	SubjectID	Grade
100	C102	A
100	C103	B
200	C102	B
300	D102	A
400	F201	C

제 3정규형(Third Normal Form)

- 이행적 종속성(Transitive dependency) 개념에 기반
  - 릴레이션 스키마 R에서, 후보 키가 아니고 어떤 키의 부분집합도 아닌 속성 집합 Z가 있을 때,
  - X → Z와 Z → Y가 만족될 때, 함수 종속 X → Y를 이행적 함수 종속이라고 부름
- Codd의 정의
  - 릴레이션 스키마 R이 2NF이고
  - R의 어떤 비주요 속성도 기본 키에 이행적으로 종속하지 않으면 R은 3NF에 속함
- 구분
  - R: 릴레이션 스키마, X: R에 속하는 릴레이션 인스턴스의 부분집합, A: R의 속성일 때
  - 다음 중 하나에 속하면 제 3 정규형에 속함
    - A ∈ X, 즉 평범한 함수 종속
    - X가 슈퍼키
    - A가 R의 어떤 키의 일부

R을 어떤 릴레이션 스키마라고 하고 X를 R에 속하는 속성들의 한 부분집합이라고 하며 A를 R의 한 속성이라고 할 때, R이 만족하는 모든 함수 종속 X → A가 다음 중 하나에 속하면 R은 제 3 정규형에 속합니다.

- A ∈ X, 즉 평범한 FD이거나, 또는
- X가 슈퍼키거나, 또는
- A가 R의 어떤 키의 일부

제 3차 정규형은 BCNF와 비슷합니다. 세 번째 항목이 다른데, 세 번째 조건을 이해하려면 릴레이션의 키라는 것들이 다른 모든 속성들을 유일하게 결정짓는 속성들의 최소 집합임을 상기해 보면 됩니다. A가 어떤 키의 일부여야 한다는 것은 키가 여러 개 있을 경우 그 중 어떤 키라도 상관없다는 뜻입니다. A가 슈퍼키의 일부인 것으로는 불충분합니다. A가 슈퍼키의 일부인 조건을 만족한다면 어떤 속성이라도 이 조건을 만족할 수 있기 때문입니다. 한 릴레이션의 스키마에서 모든 키들을 찾아내는 문제는 NP-Complete라고 알려져 있고, 또 어떤 릴레이션의 스키마가 3NF에 속하는가를 알아내는 문제도 마찬가지입니다.

어떤 종속성 X → A가 3NF를 위배하는 경우는 다음 두 가지입니다.

- X는 어떤 키 K의 진부분집합입니다. 이러한 종속성을 부분 종속성(Partial Dependency)이라고 부릅니다. 이 경우에는 (X, A) 쌍 여러 개를 중복해서 저장하게 됩니다. 아래와 같이 항공편 번호, 승객 번호, 예약 일자, 신용카드 번호를 저장하는 릴레이션 스키마가 있을 때,

```
Reservation(FlightNo, PassengerNo, Date, CreditCardNo)
              F           P         D        C
```

릴레이션 Reservation에 키는 FPD 밖에 없고 함수 종속 P → C가 존재합니다. 여기에서는 한 고객에 대한 신용카드 번호가 고객의 예약 수 만큼 존재하게 됩니다.

- X는 어떤 키의 진부분집합도 아닙니다. 이러한 종속성은 K → X → A라고 하는 연쇄적인 종속을 의미하기 때문에 이행적 함수 종속(Transitive Dependency)라고 부릅니다. 문제는 어떤 K 값에 어떤 X 값을 연관시킬 때에는 그 X값이 어떤 A값을 연관시켜 주어야 한다는 것입니다. 아래와 같은 릴레이션이 있을 때,

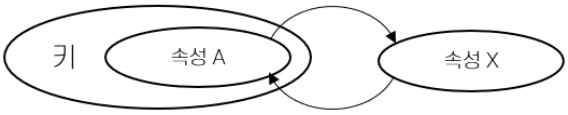
```
Parttime_emp(EmpID, Name, Parkingslot, Grade, WagePerHr, Workingtime)
               E      M         P          G        W           T
```

여기서 유일한 키는 E이고, 함수 종속 W → T가 존재하기 되므로 E → W → T라는 연쇄가 나타나게 됩니다. 따라서 직원 E가 등급 G를 가진다는 것을 기록하려면 해당 등급에 해당하는 시간당 임금을 같이 삽입해야 합니다. 이로 인해 삽입 이상, 삭제 이상, 갱신 이상이 발생합니다.

부분적 함수 종속은 다음과 같이 표시할 수 있습니다.

이행적 함수 종속은 다음과 같이 표시할 수 있습니다.

A가 키에 속하지 않는 경우



A가 키에 속하는 경우

제 3차 정규형을 사용하는 이유는 기술적인 측면이 많습니다. 키 속성과 관련한 어떤 형태의 종속성들을 특별히 취급함으로써 바람직한 어떤 성질을 준수하는 분해 작업만으로도 모든 릴레이션 스키마를 3NF 릴레이션들로 분해할 수 있게 됩니다. 이런 요건은 BCNF에는 없습니다. 3NF의 정의는 BCNF의 요건을 완화시켜서 보장이 가능하도록 합니다.

BCNF와는 달리 3NF에서는 중복성이 일부 발생할 수 있습니다. 만일 비평범 종속성 X → A가 존재하고 X가 슈퍼키가 아니라면 A가 키의 일부라서 이 릴레이션이 3NF에 속한다 하더라도 부분 종속성 및 이행 종속성에 따른 문제가 발생할 수 있습니다.

아래 릴레이션 스키마에서,

```
Reservation(FlightNo, PassengerNo, Date, CreditCardNo)
              F           P         D        C
```

함수 종속 P → C 가 존재하여 한 승객이 예약에 대한 지불을 위해 하나의 신용카드만 사용해야 한다는 조건이 있다고 할 때, P는 키가 아니고 C도 키가 아닙니다. (여기에서 키는 FPD 입니다) 따라서 이 릴레이션은 3NF가 아닙니다. 하지만 신용카드가 소유주를 유일하게 식별한다면 함수 종속 C → P를 얻게되고, CDP도 키가 됩니다. 따라서 종속성 C → P는 3NF를 위배하지 않게 되므로 Reservation 릴레이션은 3NF에 속하게 되지만, F 값이 같은 투플들은 (P, C) 쌍들을 중복해서 저장하게 됩니다.

Boyce-Code 정규형 (BCNF)

- R: 릴레이션 스키마, X: R에 속하는 릴레이션 인스턴스의 부분집합, A: R의 속성일 때
- R이 만족하는 모든 함수 종속 X → A가 다음 중 하나에 속하면 BCNF에 속함
  - A ∈ X, 즉 평범한 함수 종속
  - X가 슈퍼키

BCNF는 원래 제 3 정규형의 간단한 형태로 제안되었으나 나중에 제 3정규형보다 더 엄격하다고 밝혀진 정규형입니다.

R을 어떤 릴레이션 스키마라고 하고 X를 R에 속하는 속성들의 한 부분집합이라고 하며 A를 R의 한 속성이라고 할 때, R이 만족하는 모든 함수 종속 X → A가 다음 중 하나에 속하면 R은 BCNF에 속합니다.

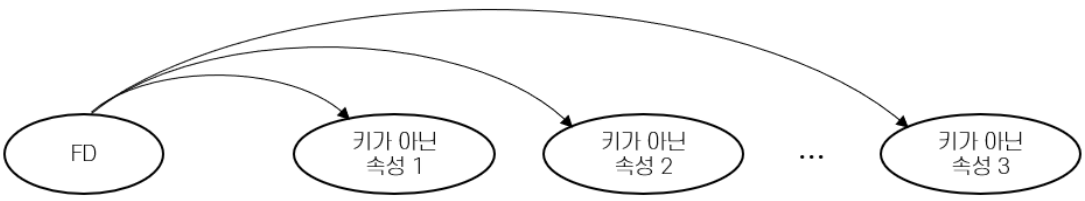
- A ∈ E, 즉 평범한 FD이거나, 또는
- X가 슈퍼키

어떤 함수 종속 집합 F가 주어졌을 때 이에 해당하는 R이 BCNF에 속하는지를 알기 위해서는, 이 정의에 따라 폐쇄 F^+에 속하는 각 X → A를 모두 따져보아야 합니다. 하지만 F에 속하는 각 종속성의 왼쪽이 슈퍼키인가를 검사하는 것 만으로 충분하다는 것을 증명할 수 있습니다.

정의에 따르면 BCNF 릴레이션에 속하는 비평범 종속성들은 당연히 키가 어떤 속성들을 결정하는 형태들뿐입니다. 따라서 이러한 릴레이션에 속하는 각 투플들은 한 키가 나머지 속성들을 묘사하는 형태의 개체이너가 관계인 것으로 볼 수 있습니다.

"모든 속성은 키(가 식별하는 개체 또는 관계)를 설명해야 하고, 키 전체를 설명하지만, 키 이외의 것은 아무것도 설명하지 않는다"

속성이나 속성의 집합을 타원형으로 표시하고 함수 종속을 화살표로 표시하면 BCNF에 속하는 릴레이션은 아래 그림과 같은 구조를 가집니다. (후보 키가 여러 개라면 후보 키 하나가 키의 역할을 담당합니다)



함수 종속 정보만 아는 경우에는 중복성의 관점에서 볼 때 BCNF가 가장 바람직한 정규형이 됩니다. 아래와 같은 릴레이션의 경우를 생각해봅시다.

X	Y	A
x	*y1*	*a*

x	*y2*	*?*

위 릴레이션 인스턴스는 속성이 X, Y, A 세 개가 있고 두 개의 투플을 가지고 있습니다. 이 두 투플은 X 필드의 값이 됩니다. 이 인스턴스가 함수 종속 X → A를 만족한다고 합시다. 그리고 중 한 투플의 A 필드의 값은 a 입니다. 이 정보로부터 두 번째 투플의 A 필드의 값도 a임을 추론해 낼 수 있습니다.

이는 실제로 a라는 값이 저장된 것입니다. 이런 상황은 BCNF에서는 일어나지 않습니다. 이 릴레이션이 BCNF에 속한다면 A와 X는 서로 다르므로 X는 키이어야 합니다. 그렇지 않다면 함수 종속 X → A는 BCNF를 위배하게 됩니다. X가 키라면 y1 = y2가 되고 두 투플의 모든 속성이 같게 됩니다. 릴레이션은 투플의 집합으로 정의되어 있으므로 동일한 투플이 두 번 나타나는 상황은 발생할 수 없습니다.

어떤 릴레이션이 BCNF에 속한다면 이 릴레이션의 한 인스턴스에 속하는 어떤 투플의 어떤 필드에 속하는 정보도 나머지 필드 정보 들로부터 함수 종속만 사용해서 유도해 낼 수 있습니다.

아래 릴레이션에서,

```
Parttime_emp(EmpID, Name, Parkingslot, Grade, WagePerHr, Workingtime)
              E     M        P          G       W           T
```

이 릴레이션인 속성이 EMPGWT이고 두 함수 종속, E → EMPGWT와 G → W을 가지고 있습니다. G가 키가 아니고 W가 키의 일부가 아니기 때문에 두 번째 함수 종속은 3NF를 위반합니다.

설계를 다시 해서, 속성이 EMGPT인 릴레이션과 속성이 GW인 릴레이션을 만들면 됩니다. EMPGT에서는 E → EMPGT가 만족되고 E가 키이며, GW에서는 G → W가 만족되고 G가 키입니다. 이 두 스키마에서 만족되는 다른 함수 종속은은 첨가 법칙으로 얻어지는 것 들 뿐입니다. 따라서 두 스키마는 모두 BCNF에 속합니다.

분해

정규화되지 않은 릴레이션은 중복성으로 인한 문제가 발생합니다. 3NF나 BCNF에 속하지 않은 릴레이션에서는 중복성으로 인한 삽입, 삭제, 갱신 문제가 발생할 수 있습니다. 이 단원에서는 릴레이션 스키마를 더 적은 속성을 가진 릴레이션 스키마로 분해하는 방법에 대해 배웁니다.

Table of Contents

분해

- 3NF나 BCNF가 아닌 릴레이션을 중복성 문제가 발생하지 않도록 처리하는 기법
- 해당 릴레이션 스키마를 더 적은 속성을 가진 릴레이션 스키마로 분해
- 릴레이션 R의 분해
  - R에 속한 속성의 부분집합으로 각기 구성된 둘 이상의 스키마들로 R을 대치
  - 이들의 속성을 모두 합쳤을 때 원래 R에 속한 속성 중에 빠지는 것이 있으면 안됨

BCNF에 속하는 릴레이션은 중복성이 없으며(FD 정보로 알 수 있는 중복성이 없다는 뜻), 3NF에 속하는 정규형도 이와 가깝습니다. 이러한 3NF나 BCNF에 속하지 않는 릴레이션에서는 종속성 때문에 문제가 발생할 수 있습니다. 이런 종속성 때문에 발생하는 문제들을 처리하는 방법은 해당 릴레이션의 스키마를 더 적은 속성을 가진 릴레이션 스키마들로 분리하는 것입니다.

릴레이션 스키마 R을 분해(Decomposition)한다는 것은 R에 속한 속성의 부분 집합으로 각기 구성된 둘 이상의 릴레이션 스키마들로 이 릴레이션 스키마를 대치한다는 것이며, 이때 이들의 속성을 모두 합쳤을 때 원래 R에 속한 속성중에 빠지는 것이 있어서는 안됩니다. 따라서 당연히 R의 주어진 인스턴스의 정보를 몇 가지로 프로젝션 해서 저장하게 됩니다.

아래 릴레이션에서,

```
Parttime_emp(EmpID, Name, Parkingslot, Grade, WagePerHr, Workingtime)
              E     M        P          G       W           T
```

이 릴레이션은 속성이 EMPGWT이고 두 함수 종속, E → EMPGWT와 G → W을 가지고 있습니다. G가 키가 아니고 W가 키의 일부가 아니기 때문에 두 번째 함수 종속은 3NF를 위반합니다.

설계를 다시 해보면 속성이 EMPGT와 GW인 릴레이션 두 개로 만들면 됩니다. EMPGT에서는 E → EMPGWT가 만족되고 E가 키가되며, GW에서는 G → W가 만족되고 G가 키가 됩니다. 이 두 스카메엇 만족되는 다른 속성들은 첨가 법칙으로 얻어지는 것뿐입니다. 따라서 이 두 스키마는 BCNF가 됩니다.

EMPGWT를 예를 들어 GMPG와 PGWT로 분해하지 않고 EMPGT와 GW로 분리하는 것은 종속성 G → W가 3NF를 위반한다는 사실을 보고 이 위배사항을 처리하는 가장 자연스러운 방법이 이 스키마로부터 W 속성을 없애는 것이라고 분석이 나왔기 때문입니다. 원래의 스키마에서 W를 없애는 대신 새 릴레이션 GW를 만들었는데, 이는 FD G → W에 의하여 모든 R은 각기 유일한 W 값과 연관되기 때문입니다.

이 시점에서 중요한 것은 릴레이션 스키마 EMPGWT의 적법한 인스턴스 R을 그에 대한 두 개의 인스턴스 EMPGT(R1)과 GW(R2)로 대치했을 때 이 R1과 R2로부터 R을 복원해 낼 수 있는 것인지 하는 것입니다. EMPGWT를 EMPGT과 GW로 분해하겠다는 것은 인스턴스 R을 R1과 R2를 나누어서 저장하는 다는 말과 같습니다. 원래 의도하던 인스턴스 R을 R1과 R2로부터 구할 수 없으면 안됩니다.

무손실 조인 분해

- R을 릴레이션 스키마라고 하고 F를 R에 대한 FD의 집합이라고 할 때,
  - R를 속성 집합 X와 속성 집합 Y의 두 스키마로 분해하였을 때
  - F에 속하는 함수 종속을 만족하는 R의 모든 인스턴스 R에 대해
  - $\pi X(r) \bowtie \pi Y(r) = R$ 이 성립하면 무손실 조인 분해라고 함

R를 한 릴레이션의 스키마라고 하고 F를 R에 대한 FD의 집합이라고 할 때, R을 속성 집합 X과 속성 집합 Y의 두 스키마로 분해하였을 때 F에 속하는 속성들을 만족하는 모든 인스턴스 R에 대해 $\pi X(r) \bowtie \pi Y(r) = R$ 이 성립하면 이를 무손실 조인 분해(lossless Join Decomposition)이라고 합니다.

이 정의는 둘 보다 많은 릴레이션으로 분해하는 경우에 대해서도 쉽게 확장할 수 있습니다. $r \subseteq \pi X(r) \bowtie \pi Y(r) = R$ 이 항상 성립한다는 건 쉽게 확인할 수 있습니다. 그렇지만 일반적으로 역은 성립하지 않습니다. 어떤 릴레이션으로부터 프로젝션을 구한 후 이들을 자연조인으로 다시 합치면 보통 원래 릴레이션에 없던 새로운 투플들이 몇 개 생기게 됩니다.

S	P	D
s1	p1	d1
s2	p2	d2
s3	p1	d3

인스턴스 r

S	P
s1	p1
s2	p2
s3	p1

$\pi SP(r)$

P	D
p1	d1
p2	d2
p1	d3

$\pi PD(r)$

S	P	D
s1	p1	d1
s2	p2	d2
s3	p3	d3
s1	p1	d3
s3	p1	d1

$\pi SP(r) \bowtie \pi PD(r)$

인스턴스 r을 인스턴스 $\pi SP(r)$와 $\pi PD(r)$로 대치하면 정보를 일부 손실하게 됩니다. 특히 r에 있는 투플들이 관계를 표현하는 경우, 관계(s1, p1, d3)와 (s3, p1, d1)이 성립하지 않는다고 더 이상 말할 수 없게 됩니다. 따라서 인스턴스 r이 적법하다면, 즉 모델링 하려고 하는 조직체에서 이러한 인스턴스가 발생할 수 있다면, 스키마 SPD를 SP과 PD로 분해하는 것은 손실성(lossy) 분해가 됩니다.

중복성을 없애기 위한 분해라면 어떠한 것이든 손실이 없는 것이어야 합니다. 다음과 같은 간단한 시험을 해 보는 것이 좋습니다.

R을 릴레이션이라고 하고 F를 R에 있는 FD들의 집합이라고 할 때, R을 속성 집합 R_1과 R_2로 구성된 두 릴레이션으로 분해하는 것이 무손실이려면 F^+에 FD $R_1 \cap R_2 \rightarrow R_1$이나 FD $R_1 \cap R_2 \rightarrow R_2$가 있어야 하며, 그 역도 성립한다.

다른 말로 하면, R1과 R2 모두에 있는 속성에는 반드시 R1이나 R2에 대한 키가 있어야 한다는 것입니다. 어떤 릴레이션을 두 개의 릴레이션으로 분해할 때에는 그 분해가 무손실- 조인 분해인지 아닌지에 대한 시험을 해 보는 것이 필요하고 충분한 조건이 됩니다.

아래 릴레이션에서,

```
Parttime_emp(EmpID, Name, Parkingslot, Grade, WagePerHr, Workingtime)
             E     M        P           G       W           T
```

이 릴레이션은 속성이 EMPGWT이고 함수 종속 G → W가 3NF를 위반합니다. 이 위반사항을 처리하려면 릴레이션을 EMPGT와 GW로 분해하면 됩니다. R이 분해된 두 릴레이션 모두에 있고 G → W가 만족되므로 이 분해는 무손실 분해입니다.

이 예를 일반화하면 다음과 같습니다.

어떤 릴레이션 R에 FD X → Y가 존재하고 X ∩ Y가 공집합이면 R을 R – Y와 XY로 무손실 분해할 수 있다.

X는 R - Y와 XY에 모두 나타나며, (X ∩ Y)가 공집합이므로), XY에 대해 키입니다. 따라서 무손실 조인 분해를 위한 검사법으로부터 위와 같은 사실에 도달할 수 있습니다.

종속성 유지 분해

- FD 프로젝션
  - R이라는 릴레이션 스키마가 있고 속성 집합 X와 Y로 된 두 개의 스키마로 분해 되며,
  - R에 대한 FD의 집합을 F라고 할 때
  - X에 속한 속성들에게만 관계되는 폐쇄 F^+에 속한 FD 집합
- 종속성 유지 분해
  - FD 집합 F를 가진 릴레이션 스키마 R을 속성 집합 X와 Y를 가진 두 스키마로 분해할 때
  - $(F_X \cup F_Y)^+ = F^+$이면 종속성 유지 분해라고 함

아래와 같은 릴레이션이 있을 때,

Contract(ContractNo, SupplierID, SubjectID, DepartmentID, PartNo, Quantity, ValuePrice)
C S J D P Q V

Contract 릴레이션의 함수 종속은 C → CSJDPQV, JP → C, SD → P가 있습니다. 이 경우 SD는 키가 아니므로 FD SD → P는 BCNF를 위반합니다.

이 위반 사항을 처리하기 위해 Contract 릴레이션을 CSJDQV와 SDP 두 릴레이션으로 분해할 수 있는데 이 분해는 무손실 조인 분해입니다. 여기에는 문제가 있는데, 무결성 제약조건 JP → C는 Contract 릴레이션에 한 투플이 삽입될 때 삽입되는 투플의 JP 값은 같지만 C 값은 다른 투플이 존재하지 않는가를 확인함으로써 쉽게 집행할 수 있습니다. 그러나 Contract 릴레이션을 CSJDQV과 SDP로 분해하고 나면 CSJDQV에 한 투플이 삽입될 때 마다 두 릴레이션을 조인해서 이 제약조건이 유지될 수 있는지를 일일이 확인해 주어야 합니다. 이런 분해를 종속성 유지 분해가 아니라고 합니다.

쉽게 말하면, 종속성 유지 분해란 투플 하나를 삽입하거나 수정할 때 하나의 릴레이션 인스턴스만 검사해보아도 FD들을 준수해 줄 수 있는 것을 말합니다. 종속성 유지 분해를 엄밀히 정의하기 위해서는 먼저 FD의 프로젝션이라는 개념을 이해해야 합니다.

R이라는 릴레이션 스키마가 있고 이것이 속성 집합 X와 Y로 된 두 개의 스키마들로 분해되며 R에 대한 FD 집합을 F라고 할 때, F의 X에 대한 프로젝션은 X에속한 속성들에만 관계되는 폐쇄 F^+에 속하는 FD의 집합입니다. 속성 X에 대한 F의 프로젝션을 F_X로 표시할 때, F^+에 속하는 종속성 U → V가 F_X에 속하려면 U와 V에 속하는 모든 속성이 X에 속한 것이어야 합니다.

FD 집합을 F를 가진 릴레이션 스키마 R을 속성 집합 X와 Y를 가진 두 스키마로 분해할 때$(F_X \cup F_Y)^+ = F^+$이면 이 분해를 **종속성 유지분해** 라고 합니다. 즉 F_X와 F_Y에 속하는 종속성들을 구해서 그들을 합집한 것의 폐쇄를 계산하면 F의 폐쇄에 속하는 모든 종속성을 얻게 됩니다. 그러므로 F_X와 F_Y에 속한 모든 종속성들만 집행하면 F^+에 속한 모든 FD들도 만족한다는 것을 알 수 있습니다. FX를 집행하려면 릴레이션 X에 대한 삽입만 따져보면 됩니다.

F의 프로젝션을 구할 때 폐쇄 F+를 고려해야 하는 이유를 알기 위해, 속성 ABC를 가진 릴레이션 R이 속성 AB와 속성 BC의 두 릴레이션으로 분해된다고 가정합시다. R에 대한 FD의 집합 F에는 A → B, B → C, C → A가 들어있습니다. 물론 A → F는 F_{AB}에 속하고 B → C는 F_{BC}에 속합니다.

F의 폐쇄에는 F자체에 속한 모든 종속성에 대하여 A → C, B → A, C → B도 포함합니다. 따라서 F_{AB}는 B → A도 포함하며 F_{BC}는 C → B도 포함합니다. 그러므로 $F_{AB} \cup F_{BC}$에는 A → B, B → C, B → A, C → B도 들어있습니다. 따라서 F_{AB}와 F_{BC}에 속한 종속성 폐쇄에는 C → B, B → A 이행 법칙에 의거하여 C → A도 들어있게 됩니다. 그러므로 이 분해는 종속성 C → A를 그대로 유지합니다.

BCNF로 분해

- R이 BCNF에 속하지 않을 때,
  - X ⊂ R이고 A는 R에 속하는 단일 속성이며 X → A 가 BCNF를 위배하는 FD라고 하면, R을 R – A와 XA로 분해
  - R – A나 XA가 BCNF에 속하지 않으면 순환해서 반복

릴레이션 스키마 R을 몇 개의 BCNF 릴레이션 스키마들로 분해하는 알고리즘은 아래와 같습니다.

1. R이 BCNF에 속하지 않는다고 할 때, X ⊂ R이고 A는 R에 속하는 단일 속성이며 X → A가 BCNF를 위배하는 FD라고 하면, R을 R – A와 XA로 분해한다.
2. R – A나 XA가 BCNF가 속하려 하지 않으면, 이 알고리즘을 순환하여 계속 적용한다.

R – A는 R에서 A에 속하지 않은 속성들의 집합을 표시한 것이고 XA는 X에 속하는 속성과 A에 속하는 속성들의 합집합을 표시한 것입니다. X → A가 BCNF를 위반하기 때문에 이 종속성은 평범한 종속성이 아입니다. 또 A는 단일 속성입니다. 그러므로 A는 X에 속하지 않고 X ∩ A는 공집합입니다. 따라서 단계 1의 분해는 무손실 조인 분해입니다.

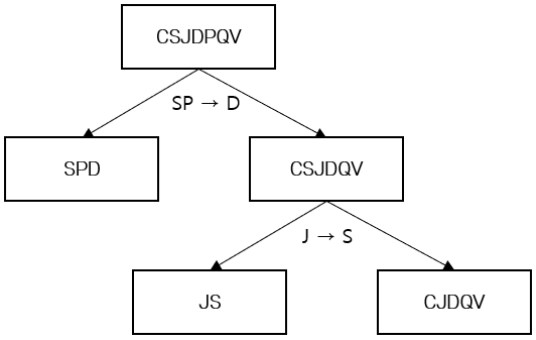
R – A와 XA에 관한 종속성의 집합은 F를 그들의 속성에 대해 프로젝션한 것과 같습니다. 이렇게 새로 얻은 릴레이션 중에 하나다 BCNF에 속하지 않으면 단계 2에서 계속해서 분해합니다. 릴레이션을 분해하게 되면 속성의 수가 계속 줄게 되므로 이 과정은 반복을 통해 종료되게 되고 결국 BCNF에 속하는 몇 개의 릴레이션 스키마들을 얻게 됩니다. 또한 이 알고리즘을 통해 얻은 릴레이션의 인스턴트들을 조인하면 원래 릴레이션에 해당하는 인스턴스를 그대로 얻게 됩니다.

아래와 같은 릴레이션이 있을 때,

```
Contract(ContractNo, SupplierID, SubjectID, DepartmentID, PartNo, Quantity, ValuePrice)
            C           S           J           D          P        Q          V
```

Contract 릴레이션의 함수 종속은 C → CSJDPQV, JP → C, SD → P가 있습니다. 이 경우 SD는 키가 아니므로 FD SD → P는 BCNF를 위반합니다. 함수 종속 SD → P를 분해의 기준으로 삼으면 두 개의 스키마 SDP과 CSJDQV를 얻습니다. SDP는 BCNF에 속합니다. 그런데 각 과제마다 공급자가 하나 뿐이라는 제약조건이 있다고 가정합시다. 이 함수 종속은 J → S 가 됩니다. 이렇게 되면 스키마 CSJDQV는 BDNF에 속하지 못합니다. 이 스키마를 J → S를 기준으로 분해하여 JS와 CJDQV로 또 분해합니다. C → JDQV는 CJDQV 상에서 만족합니다. 만족하는 다른 FD이라고는 이 FD에 첨가 법칙을 적용하여 얻는 것들 뿐이며 이러한 모든 FD의 왼편에는 키가 들어있습니다. 그러므로 스키마 SDP, JS, CJDQV는 모두 BCNF에 속하도 이 스키마의 모임은 CDJDQV의 무손실 조인 분해를 나타냅니다.

이런 분해 과정상의 단계들을 아래와 같이 트리 형태로 나타낼 수 있습니다.


CSJDPQV
SP → D
SPD
CSJDQV
J → S
JS
CJDQV

검토

- 스키마 정제 개요
- 정규화와 정규형
- 분해