데이터베이스: 강의노트 06

A. Silberschatz, H. Korth, S. Sudarshan
Database System Concepts,
Fourth Edition, McGraw-Hill, 2002.

Part II. Relational Databases

7 관계형 데이터베이스의 설계

- 목표
  - 불필요하게 중복 저장되는 것을 줄이는 것. (중복이 많으면 저장 공간이 낭비되며, 비일관성이 초래될 확률이 높아진다.)
  - 정보 검색을 효율적으로 할 수 있게 스키마를 정의하는 것
- 이 목표를 충족시키기 위한 방법 중 하나는 **정규형(normal form)**에 맞게 스키마를 정의하는 것이다.
- **정규화(normalization)**란 데이터를 데이터베이스에 효과적으로 조직화하는 과정으로, 정규형은 이 과정에서 참고할 수 있는 지침이다.

7.1 제1 정규형

- 관계 스키마 R의 모든 속성의 도메인이 원자적이면 R은 제1 정규형(1NF)에 속한다고 한다.
- 복합 속성(예: 주소)과 다중값 속성(예: 전화번호)은 원자적 속성이 아니다.
- 속성의 도메인이 원자적이 아닐 때 발생할 수 있는 문제
  - 복합 속성에서 세부 필드를 추출하는 부가적인 프로그래밍이 필요할 수 있다.
  - 데이터 중복의 가능성: 예) *depositor* 관계 대신에 *customer* 관계에 계좌의 집합이라는 다중값 속성을 추가하고, *account* 관계에 고객의 집합이라는 다중값 속성을 추가했을 경우
- 도메인을 실제 데이터베이스에 어떻게 사용하느냐가 중요하다.
- 예) 직원의 식별번호를 CS0012처럼 부서 식별자(CS)와 부서 내의 직원의 고유번호(0012)를 합쳐 만든다고 하자. 이 경우 이 도메인은 원자적 속성이 아니다. 이와 같은 경우에 직원이 소속된 부서를 알고 싶으면 식별번호에서 앞 두 문자를 추출하는 부가적인 프로그래밍이 필요하게 된다. 이것은 데이터베이스에서 이루어지지 않고 응용에서 이루어지게 된다. 뿐만 아니라 이것을 주키로 사용하면 직원이 부서를 옮길 때 직원이 나타나는 모든 곳을 수정해야 한다.
- 오히려 비원자적 속성이 유용할 수도 있다. 그렇기 때문에 데이터베이스에서 복합 속성을 제공한다.

7.2 관계형 데이터베이스 설계시 함정

- 잘못 설계로 인해 발생하는 문제점
  - 데이터의 중복
  - 특정 정보의 표현 불가능
- 예) *loan, borrower, branch* 관계를 사용하는 대신에 다음과 같은 하나의 관계만을 사용한다고 하자.

  Lending-schema = (지점명,지점-도시,자산, 고객명,대출번호,대출액)

  - 문제 1. 지점과 관련된 정보가 중복된다. 저장 공간이 낭비되고 있으며, 지점 정보의 갱신이 어렵다. 예) 지점의 자산 정보의 변경
  - 이 예에서 지점이 주어지면 그것의 자산은 독특하게 결정된다. 그러나 한 지점에 여러 대출이 있을 수 있으므로 지점은 대출을 결정할 수 없다. 따라서 지점명과 자산은 **함수 종속(functional dependency)**이 있다고 하며, 다음과 같이 표현한다.

    $$\text{지점명} \rightarrow \text{자산}$$

    하지만 지점명과 대출번호는 함수 종속 관계가 없다. 종속 관계가 없는 것은 다른 관계를 통해 나타내는 것이 바람직하다. 함수 종속을 다른 말로 유일값 제약조건(unique value constraint)이라 한다. 즉, 함수 종속은 관계가 충족해야 하는 제약조건 중 하나이다.
  - 문제 2. 대출이 없는 지점에 관한 정보를 직접 나타낼 수 없다. 널 값을 사용할 수 있지만 널 값은 처리하기가 어렵다.

7.3 함수 종속

7.3.1 기본 개념

정의 7.1 (함수 종속). 관계 스키마 R, $\alpha \subseteq R$, $\beta \subseteq R$이 있을 때, 모든 적법한 관계 $r(R)$에서 $t_1[\alpha] = t_2[\alpha]$을 만족하는 r에 속한 모든 투플 쌍 t_1과 t_2에 대해 $t_1[\beta] = t_2[\beta]$이면 다음과 같은 함수 종속이 관계 스키마 R에 성립한다고 한다.

$$\alpha \rightarrow \beta$$

□

정의 7.2 (수퍼키). 관계 스키마 R과 $K \subseteq R$가 있을 때, $K \rightarrow R$이 성립하면 K는 R의 수퍼키이다. □

- 함수 종속은 다음 두 가지 용도로 사용한다.
  - 어떤 관계가 주어진 함수 종속 집합을 만족하는지 검사하기 위한 용도로 사용한다. 관계 r이 주어진 함수 종속 집합 F를 만족하면 r은 F를 충족한다고 말한다.
  - 주어진 관계에 제약조건을 명시하기 위한 용도로 사용한다. 함수 종족 집합 F를 충족하는 관계 스키마 R의 관계만 다루고자 하면 F는 R에서 성립한다고 말한다.
- 예1) 다음과 같은 관계 r이 있다고 하자.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

  - $A \rightarrow C$는 충족된다. 그러나 $C \rightarrow A$는 충족되지 않는다.
  - $AB \rightarrow D$도 충족된다.
- 모든 관계에서 충족되는 함수 종속은 **자명(trivial)**하다고 말한다. 예) $A \rightarrow A$, $AB \rightarrow A$
- $\beta \subseteq \alpha$이면 함수 종속 $\alpha \rightarrow \beta$은 자명하다.
- 예2) *customer* 관계
  - '고객-거리 → 고객-도시' 함수 종속을 충족한다. 그러나 보통 서로 다른 도시에 같은 거리명이 있을 수 있으므로 '고객-거리 → 고객-도시' 함수 종속은 *customer* 관계에서 성립하지 않는다.
  - **중요**. 어떤 관계가 어떤 함수 종속을 충족하고 있어도 그 함수 종속이 그 관계에 성립한다고 말할 수는 없다.
- 예3) *loan* 관계
  - '대출번호 → 대출액' 함수 종속을 충족한다. 또한 '대출번호 → 대출액' 함수 종속은 *loan* 관계에서 성립한다.
- 은행 데이터베이스에 성립해야 하는 함수 종속
  - *Branch-schema*

    지점명 → 지점-도시
    지점명 → 자산
  - *Customer-schema*

    고객명 → 고객-도시
    고객명 → 고객-거리
  - *Loan-schema*

    대출번호 → 대출액
    대출번호 → 지점명
  - *Borrower-schema*: 없음
  - *Account-schema*

    계좌번호 → 잔액
    계좌번호 → 지점명
  - *Depositor-schema*: 없음

7.3.2 함수 종속 집합의 닫힘

- 함수 종속 집합 F을 충족하는 모든 관계 $r(R)$이 관계 스키마 R에 대한 함수 종속 f를 충족하면 f는 F에 논리적으로 내포되어 있다고 한다.
- 예1) 관계 스키마 $R = (A, B, C, G, H, I)$과 R에서 성립하는 다음과 같은 함수 종속 집합이 있다고 하자.

$$\begin{aligned} A &\rightarrow B \\ A &\rightarrow C \\ CG &\rightarrow H \\ CG &\rightarrow I \\ B &\rightarrow H \end{aligned}$$

  그러면 이 함수 종속 집합은 $A \rightarrow H$를 논리적으로 내포하고 있다.
- F의 **닫힘(closure)**이란 F에 의해 논리적으로 내포하고 있는 모든 함수 종속의 집합을 말하며, F^+로 나타낸다.
- Armstrong은 F^+를 구할 때 사용할 수 있는 다음과 같은 공리(axiom) 또는 추론 규칙을 제안하였다.
  - **반사 규칙(reflexivity rule)**: α가 속성의 집합이고 $\beta \subseteq \alpha$가 성립하면 $\alpha \rightarrow \beta$가 성립한다.
  - **확대 규칙(augmentation rule)**: $\alpha \rightarrow \beta$가 성립하고 γ가 속성의 집합이면 $\gamma\alpha \rightarrow \gamma\beta$가 성립한다.
  - **추이 규칙(transitivity rule)**: $\alpha \rightarrow \beta$와 $\beta \rightarrow \gamma$가 성립하면 $\alpha \rightarrow \gamma$가 성립한다.

  Armstrong의 공리는 건전(sound)하며 완전(complete)하다. 즉, 잘못된 함수 종속을 만들어 내지 않으며, 이 공리들을 이용하면 모든 F^+을 생성할 수 있다. 그러나 Armstrong의 공리를 적용하여 F^+을 생성하는 것은 불편하다. 따라서 보통 다음을 규칙을 사용한다.
  - **결합 규칙(union rule)**: $\alpha \rightarrow \beta$와 $\alpha \rightarrow \gamma$가 성립하면 $\alpha \rightarrow \beta\gamma$가 성립한다.
  - **분해 규칙(decomposition rule)**: $\alpha \rightarrow \beta\gamma$가 성립하면 $\alpha \rightarrow \beta$와 $\alpha \rightarrow \gamma$가 성립한다.
  - **의사추이 규칙(pseudotransitivity rule)**: $\alpha \rightarrow \beta$와 $\gamma\beta \rightarrow \delta$가 성립하면 $\alpha\gamma \rightarrow \delta$가 성립한다.

$F^+ = F$
F^+가 변하지 않을 때까지 다음을 반복
    F^+에 있는 각 함수 종속 f에 대해
        f에 반사규칙과 확대규칙을 적용
        새 함수 종속을 F^+에 추가
    F^+에 있는 각 함수 종속 쌍 f_1, f_2에 대해
        추이규칙을 적용
        새 함수 종속을 F^+에 추가

<그림 7.1> F^+를 계산하는 알고리즘

result := α
result가 변할 때까지 다음을 반복o
    각 $\beta \rightarrow \gamma \in F$에 대해
        $\beta \subseteq$ result이면 result:=result $\cup \gamma$;

<그림 7.2> α^+를 계산하는 알고리즘

- Armstrong의 공리를 이용하여 F^+를 계산하는 알고리즘: 그림 7.1
- Armstrong의 알고리즘을 이용하여 F^+를 계산하는 것은 쉽지 않다.

7.3.3 속성 집합의 닫힘

- α가 속성들의 집합이고, 함수 종속 집합 F가 주어졌을 때, F에 대한 α의 닫힘이란 F에 따라 α에 의해 결정되는 모든 속성의 집합을 말하며, α^+로 나타낸다.
- α^+를 계산하는 알고리즘: 그림 7.2
  - $\beta \subseteq$ result이면 반사 규칙에 의해 result $\rightarrow \beta$가 성립한다. 따라서 $\beta \rightarrow \gamma$가 성립하면 추이 규칙에 의해 result $\rightarrow \gamma$가 성립한다.
- 예1) $(AG)^+$의 계산
  - result $= AG$
  - $A \rightarrow B$ 때문에 result $= ABG$
  - $A \rightarrow C$ 때문에 result $= ABCG$
  - $CG \rightarrow H$ 때문에 result $= ABCGH$
  - $CG \rightarrow I$ 때문에 result $= ABCGHI$
- 속성의 닫힘 알고리즘의 용도
  - **수퍼키의 검사**: α^+를 계산하여, 이 집합에 모든 R의 속성이 포함되면 α는 R의 수퍼키가 된다.
  - 함수 종속 $\alpha \rightarrow \beta$의 성립여부 검사: α^+에 β가 포함되면 $\alpha \rightarrow \beta$는 성립한다.
  - F^+를 계산하는 방법: 각 $\gamma \subseteq R$에 대해 γ^+를 계산하고, 각 $S \subseteq \gamma$에 대해 함수 종속 $\gamma \rightarrow S$를 F^+에 추가한다. 이 방법을 이용하여 F^+를 계산하는 것 역시 어렵다.

7.4 정규 커버

- 함수 종속 집합 F가 성립하는 관계 스키마 R에 대한 관계 $r(R)$이 있을 때, 이 관계에 대한 갱신은 F를 충족해야 한다.
- 이 검사를 쉽게 하기 위해 불필요한 함수 종속을 F에서 제거한다.
- 주어진 함수 종속 집합 F에 속한 어떤 함수 종속 $\alpha \rightarrow \beta$에서 $A \in \alpha$ 또는 $A \in \beta$를 제거하여도 F^+가 변하지 않으면 A는 **여분 속성(extraneous attribute)**이라 한다.
- 여분 속성의 정의
  - $A \in \alpha$에 대해 F가 $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$를 논리적으로 내포하고 있으면 A는 α에서 여분 정보이다.
  - $B \in \beta$에 대해 $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\}$가 F를 논리적으로 내포하고 있으면 B는 β에서 여분 정보이다.
- 예1) F에 $AB \rightarrow C$와 $A \rightarrow C$가 있으면 $AB \rightarrow C$에서 B는 여분 속성이다.
- 예2) F에 $AB \rightarrow CD$와 $A \rightarrow C$가 있으면 $AB \rightarrow CD$에서 C는 여분 속성이다.
- 함수 종속 $\alpha \rightarrow \beta$에서 여분 속성 여부를 효율적으로 검사하는 방법
  - $A \in \beta$에 대해서는

    $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$$

    을 계산한 다음 F'으로부터 $\alpha \rightarrow A$를 유추할 수 있는지 검사한다. 이를 위해 F'에 대한 α^+를 계산한 다음에 A가 α^+에 포함되는지 확인한다. 만약 포함되면 A는 β에서 여분 속성이다.
  - $A \in \alpha$에 대해서는 먼저 $\gamma = \alpha - \{A\}$를 계산하고 $\gamma \rightarrow \beta$를 F로부터 유추할 수 있는지 검사한다. 이를 위해 F에 대한 γ^+를 계산한 다음에 γ^+에 β의 모든 속성이 포함되는지 확인한다. 만약 포함되면 A는 α에서 여분 속성이다.
- 예3) F에 $AB \rightarrow CD$, $A \rightarrow E$, $A \rightarrow C$가 포함되어 있을 때 $AB \rightarrow CD$에서 C는 여분 속성인가?
  - $F' = \{AB \rightarrow D, A \rightarrow E, A \rightarrow C\}$
  - $(AB)^+ = ABCDE$이므로 C는 $AB \rightarrow CD$에서 여분 속성이다.
- 예4) F에 $AB \rightarrow CD$, $A \rightarrow E$, $A \rightarrow C$가 포함되어 있을 때 $AB \rightarrow CD$에서 D는 여분 속성인가?
  - $F' = \{AB \rightarrow C, A \rightarrow E, A \rightarrow C\}$
  - $(AB)^+ = ABCE$이므로 D는 $AB \rightarrow CD$에서 여분 속성이 아니다.

$F_c = F$
F_c가 변하지 않을 때까지 다음을 반복
  결합 규칙을 사용하여
  다음과 같은 규칙을 결합한다.
    $\alpha_1 \rightarrow \beta_1, \alpha_1 \rightarrow \beta_2$를 $\alpha_1 \rightarrow \beta_1\beta_2$로
  각 종속에 여분 속성이 있는지 검사하고
  있으면 제거한다.

<그림 7.3> 정규 커버를 계산하는 알고리즘

- F에 대한 **정규 커버(canonical cover)** F_c는 종속들의 집합으로서 F는 논리적으로 F_c에 있는 모든 종속을 내포하고 있으며, F_c도 논리적으로 F에 있는 모든 종속을 내포하고 있어야 한다. 또한 다음 두 특성을 만족해야 한다.
  - F_c에는 여분 속성이 있는 함수 종속이 없어야 한다.
  - F_c의 모든 함수 종속에서 좌측항은 유일해야 한다. 즉, F_c에 두 종속 $\alpha_1 \rightarrow \beta_1$과 $\alpha_2 \rightarrow \beta_2$가 있을 때 $\alpha_1 \neq \alpha_2$이어야 한다.
- 정규 커버를 계산하는 알고리즘: 그림 7.3
- 예5) 관계 스키마 (A, B, C)에서 성립하는 함수 종속 집합 F가 다음과 같다.

$$\begin{aligned} A &\rightarrow BC \\ B &\rightarrow C \\ A &\rightarrow B \\ AB &\rightarrow C \end{aligned}$$

  F의 정규 커버 F_c를 구하라.
  - 단계 1. $A \rightarrow BC$와 $A \rightarrow B$를 결합하면 $A \rightarrow BC$가 된다. 결과 F_c는 다음과 같다.

$$\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$$

  - 단계 2. $AB \rightarrow C$에서 A가 여분 속성인지 검사하여 보자. F_c에서 $B^+ = \{B, C\}$이므로 A는 여분 속성이다. 결과 F_c는 다음과 같다.

$$\{A \rightarrow BC, B \rightarrow C\}$$

  - 단계 3. $A \rightarrow BC$에서 C는 여분 속성인지 검사하여 보자. $F' = \{A \rightarrow B, B \rightarrow C\}$에서 $A^+ = \{B, C\}$이므로 C는 여분 속성이다. 따라서 최종 F_c는 다음과 같다.

$$\{A \rightarrow B, B \rightarrow C\}$$

- 예6) 관계 스키마 (A, B, C)에서 성립하는 함수 종속 집합 F가 다음과 같다.

$$\begin{aligned} A &\rightarrow BC \\ B &\rightarrow AC \\ C &\rightarrow AB \end{aligned}$$

  F의 정규 커버 F_c를 구하라.

지점명	지점-도시	자산	고객명	대출 번호	대출액
강남점	서울	9000	유비	L-17	1000
천안점	천안	3100	장비	L-23	2000
상록점	안산	2700	관우	L-15	1500
강남점	서울	9000	조운	L-14	1500
천안점	천안	3100	유비	L-93	500

<그림 7.4> *lending* 관계

  - 단계 1. 결합 법칙을 적용할 규칙은 없다.
  - 단계 2. $A \rightarrow BC$에서 B와 C는 모두 여분 속성이다. 하지만 둘을 모두 제거하는 것은 옳지 않다. B를 제거하면 결과 F_c는 다음과 같다.

$$\{A \rightarrow C, B \rightarrow AC, C \rightarrow AB\}$$

    여기서 C는 여분 속성이 아니다.
  - 단계 3. $B \rightarrow AC$에서 A와 C는 모두 여분 속성이다. A를 제거하면 F_c는 다음과 같다.

$$\{A \rightarrow C, B \rightarrow C, C \rightarrow AB\}$$

    $C \rightarrow AB$에서 A와 B는 모두 여분 속성이 아니다. 각 단계에서 다른 것을 삭제하였으면 다른 정규 커버를 얻을 수 있다. 즉, 정규 커버는 유일하지 않다.

7.5 분해

- 7.2 절의 제시된 *Lending-schema*를 통해 알 수 있듯이 많은 수의 속성을 가진 하나의 관계를 사용하는 것보다는 적은 수의 속성을 가지는 여러 관계로 분해하여 사용하는 것이 바람직하다. 하지만 잘못 분해하면 오히려 역효과가 발생한다.
- 예) *lending*(*Lending-schema*) 관계가 그림 7.4과 같다고 하자. 이 때 *Lending-schema*를 다음과 같은 두 관계 스키마로 분해하였다고 하자.

  Branch-customer-schema =
    (지점명,지점-도시,자산,고객명)
  Customer-loan-schema =
    (고객명,대출번호,잔액)

  branch-customer(*Branch-customer-schema*)
  customer-loan(*Customer-loan-schema*)

  그러면 각 관계는 다음과 같다.

  branch-customer

지점명	지점-도시	자산	고객명
강남점	서울	9000	유비
천안점	천안	3100	장비
상록점	안산	2700	관우
강남점	서울	9000	조운
천안점	천안	3100	유비

  customer-loan

지점명	지점-도시	자산	고객명	대출번호	대출액
강남점	서울	9000	유비	L-17	1000
강남점	서울	9000	유비	L-93	500
천안점	천안	3100	장비	L-23	2000
상록점	안산	2700	관우	L-15	1500
강남점	서울	9000	조운	L-14	1500
천안점	천안	3100	유비	L-93	500
천안점	천안	3100	유비	L-17	1000

<그림 7.5> *branch-customer* ⋈ *customer-loan* 관계

고객명	대출번호	대출액
유비	L-17	1000
장비	L-23	2000
관우	L-15	1500
조운	L-14	1500
유비	L-93	500

대출액이 800 이하인 대출을 가지고 있는 모든 지점을 찾아야 할 경우에는 두 관계를 조인해야 한다.

branch-customer ⋈ *customer-loan*

두 관계를 조인한 결과는 그림 7.5와 같다. 조인한 결과를 보면 원래는 없었던 다음과 같은 새로운 투플이 존재한다.

(강남점,서울,9000,유비,L-93,500)
(천안점,천안,9000,유비,L-17,1000)

따라서 질의 결과에 포함되지 않아야 하는 "강남점"이 포함된다. 즉, 고객의 대출이 어느 지점에서 대출된 것인지 알 수 없다. 다시 말하면 보다 많은 투플을 얻게 되지만 정보의 손실이 있다. 그러므로 이와 같은 분해를 **손실 분해(lossy decomposition)** 또는 **손실 조인 분해(lossy-join decomposition)**이라 한다. 반대로 손실이 없는 분해를 **무손실 조인 분해(lossless-join decomposition)**이라 한다.

- R은 관계 스키마이고, 다음이 성립하면

$$R = R_1 \cup R_2 \cup \cdots \cup R_n$$

$\{R_1, R_2, \ldots, R_n\}$은 R의 분해라 한다.

- $r(R)$이고 $r_i = \Pi_{R_i}(r)$이면 다음은 항상 성립한다.

$$r \subseteq r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

하지만 일반적으로 다음이 성립한다.

$$r \neq r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

만약 다음이 성립하면 이 분해는 무손실 조인 분해이다.

$$r = r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

7.6 분해의 바람직한 특성

- 예) *Lending-schema*에서 만족되어야 할 함수 속성은 다음과 같다.

지점명 → 지점-도시 자산
대출번호 → 대출액 지점명

이 관계 스키마를 다음과 같이 분해한다.

Branch-schema = (지점명,지점-도시,자산)
loan-schema = (대출번호,지점명,잔액)
Borrower-schema = (고객명,대출번호)

이 분해는 이미 기존 장에서 사용하던 관계 스키마이다.

7.6.1 무손실 조인 분해

- 분해는 반드시 무손실 조인 분해이어야 한다.
- 분해가 무손실 조인 분해임을 검사하는 방법
  - R이 관계 스키마이고, F는 R에서 성립하는 함수 종속 집합이다. 이 때 R_1과 R_2가 R의 분해이면 최소한 다음 중 하나가 F^+에 존재해야만 무손실 조인 분해가 된다.
    - $R_1 \cap R_2 \rightarrow R_1$
    - $R_1 \cap R_2 \rightarrow R_2$

    즉, $R_1 \cap R_2$는 R_1 또는 R_2의 수퍼키이어야 한다. 수퍼키가 아닌 것을 기준으로 조인하면 중복되는 투플이 있을 수 있다.
- 예) *Lending-schema*를 다음 두 관계 스키마로 분해하였다고 하자.

Branch-schema = (지점명,지점-도시,자산)
loan-info-schema =
(지점명,고객명,대출번호,잔액)

두 관계 스키마의 교집합은 지점명이며, 지점명은 *Branch-schema*의 수퍼키이므로 이 분해는 무손실 조인 분해이다. 그 다음 *loan-info-schema*를 다음 두 관계 스키마로 분해하였다고 하자.

loan-schema = (대출번호,지점명,잔액)
Borrower-schema = (고객명,대출번호)

두 관계 스키마의 교집합은 대출번호이며, 대출번호는 *loan-schema*의 수퍼키이므로 이 분해 또한 무손실 조인 분해이다.

7.6.2 종속 유지

- 데이터베이스를 갱신할 때 함수 종속은 유지되어야 한다. 따라서 이 검사를 효율적으로 하기 위해서는 조인을 하지 않아도 이 검사를 할 수 있어야 한다.
- F가 관계 스키마 R에서 성립하는 함수 종속의 집합이고, R_1, R_2, ..., R_n이 R의 분해라 하자. R_i에 대한 F의 제한(restriction) F_i은 F^+에 있는 모든 함수 종속 중 R_i의 속성만으로 구성된 함수 종속 집합을 말한다.

compute F^+
각 $R_i \in D$에 대해 F_i를 계산
$F' = F_1 \cup F_2 \cup \cdots \cup F_n$
compute F'^+
$F'^+ \stackrel{?}{=} F^+$

<그림 7.6> 종속 유지 여부를 검사하는 알고리즘 A

각 $\alpha \rightarrow \beta \in F$에 대해
result = α
result가 변하지 않을 때까지 다음을 반복
  각 $R_i \in D$에 대해 다음을 계산
    $t = (\text{result} \cap R_i)^+ \cap R_i$
    result = result $\cup\, t$
$\beta \subset$ result이면 $\alpha \rightarrow \beta$는 유지된다.

<그림 7.7> 종속 유지 여부를 검사하는 알고리즘 B

- 제한의 집합 $F_1, F_2, \ldots, F_n$은 효율적으로 검사할 수 있는 종속 집합들이다. 따라서 이것만 검사하는 것이 충분하면 함수 종속이 분해에 유지되는 것이다. 즉, **함수 종속이 분해에 유지된다는 것은 분해를 하여도 조인 없이 그 함수 종속을 확인할 수 있다는 것**을 말한다.
- $F' = F_1 \cup F_2 \cup \cdots \cup F_n$일 때, 일반적으로 $F' \neq F$이다. $F' \neq F$이어도 $F'^+ = F^+$가 성립할 수 있다. 만약 $F'^+ = F^+$가 성립하면 이 분해는 **종속 유지 분해(dependency-preserving decomposition)**이다.
- R의 분해 $D = \{R_1, R_2, \ldots, R_n\}$에 대해 종속 유지 여부를 검사하는 알고리즘: 그림 7.6
- 보다 효율적인 검사 방법(F^+를 계산할 필요가 없음): 그림 7.7
  - 이 알고리즘에서 속성 닫힘은 F와 관련되어 계산한다.

7.6.3 정보의 중복

- 정보의 중복은 최대한 피하는 것이 바람직하다. 따라서 분해에서도 정보의 중복을 없애는 것이 중요하다.
- 중복을 줄이는 방법은 스키마가 다음에 설명되는 각 종 정규형에 속하도록 하는 것이다.

7.7 BCNF

7.7.1 정의

정의 7.3 (BCNF, Boyce-Codd Normal Form). 함수 종속 집합 F에 대해 관계 스키마 R이 다음을 만족하면 BCNF에 속한다고 한다.
F^+에 있는 각 함수 종속 $\alpha \rightarrow \beta$은 다음 중 하나를 만족해야 한다. 여기서 $\alpha \subseteq R$이고, $\beta \subseteq R$이다.

- $\alpha \rightarrow \beta$는 자명한 함수 속성이다.
- α는 R의 수퍼키이다.

□

- 데이터베이스 설계가 BCNF에 속한다는 것은 이 데이터베이스의 모든 관계 스키마가 BCNF에 속한다는 것을 의미한다.
- 예1) *Customer-schema*와 그것의 함수 종속

  Customer-schema =
  (고객명,고객-거리,고객-도시)
  고객명 → 고객-도시 고객-거리

  이 스키마는 BCNF에 속한다.
- 예2) *Loan-info-schema*와 그것의 함수 종속

  Loan-info-schema =
  (지점명,고객명,대출번호,대출액)
  대출번호 → 대출액 지점명

  대출번호는 이 스키마의 수퍼키가 아니므로 이 스키마는 BCNF에 속하지 않는다. 또한 함수 종속 '대출번호 → 대출액'은 자명한 함수 종속이 아니다.
- 관계 $r(R)$이 BCNF를 충족하는지 검사하는 방법
  - 자명하지 않는 함수 종속 $\alpha \rightarrow \beta$가 BCNF를 위배하는지 검사한다. 이를 위해 α^+를 계산하고, α^+에 R의 모든 속성이 포함는지 검사한다. 즉, α가 R의 수퍼키임을 검사한다.
  - F^+ 대신에 F에 대해서만 BCNF를 위배하는지 검사하는 것으로 R이 BCNF에 속하는지 알 수 있다.
- R의 분해 R_i가 BCNF에 속하는지 검사할 때에는 F에 대한 검사만으로는 충분하지 않다.
- 예3) $R(A, B, C, D, E)$, $F = \{A \rightarrow B, BC \rightarrow D\}$
  - R을 $R_1(A, B)$, $R_2(A, C, D, E)$로 분해하였을 경우 F 중에 R_2의 속성으로만 구성된 종속이 없어 R_2가 BCNF를 충족한다고 생각할 수 있다.
  - 의사 추이 규칙을 적용하면 $AC \rightarrow D \in F^+$이다. 이 종속은 자명하지 않으며, AC는 수퍼키가 아니므로 R_2는 BCNF에 속하지 않는다.
- R의 분해 R_i가 BCNF에 속하는지 검사하는 방법
  - R_i의 모든 부분집합 α에 대해 α^+를 계산하고 각 닫힘에 $R_i - \alpha$에 어떤 속성도 포함되지 않거나 모두 포함되면 R_i는 BCNF에 속한다.

7.7.2 분해 알고리즘

- BCNF 분해 알고리즘: 그림 7.8
- 이 알고리즘은 BCNF로 분해해줄 뿐만 아니라 무손실 조인 분해를 해준다.

result := $\{R\}$;
done := false;
compute F^+;
done이 참이 될 때까지
  result에 있는 R_i 중에 BCNF에 속하지
  않는 것이 있으면
    $\alpha \rightarrow R_i \notin F^+$이고 $\alpha \cap \beta = \emptyset$인
    $\alpha \rightarrow \beta$가 R_i에서 성립하는
    자명하지 않는 함수 종속이면
    result := (result $- R_i$) $\cup$ $(R_i - \beta) \cup (\alpha, \beta)$;
  없으면 done := true;

<그림 7.8> BCNF 분해 알고리즘

- 예) *Lending-schema*와 그것의 함수 종속

  Lending-schema = (지점명,지점-도시,자산,
  고객명,대출번호,대출액)
  지점명 → 자산 지점-도시
  대출번호 → 대출액 지점명

  이 스키마의 후보키는 (대출번호,고객명)이다.

  – 지점명은 이 스키마의 수퍼키가 아니므로 이 스키마는 BCNF에 속하지 않는다.
  – 따라서 다음과 같이 분해한다.

    Branch-schema =
    (지점명,지점-도시,자산)
    Loan-info-schema =
    (지점명,고객명,대출번호,대출액)

    여기서 *Branch-schema*는 BCNF에 속하지만 *Loan-info-schema*에서 대출번호는 수퍼키가 아니므로 다시 이 스키마를 다음과 같이 분해한다.

    Loan-schema =
    (대출번호,지점명,대출액)
    Borrower-schema = (고객명,대출번호)

  위 두 스키마는 모두 BCNF에 속한다.

7.7.3 종속 유지

- BCNF 분해는 항상 종속을 유지하지는 않는다.
- 예1) 다음과 같은 은행가 스키마와 그 스키마에서 성립하는 함수 종속이 있다고 하자. 즉, 고객은 특정 지점에 그 고객을 담당하는 개인 은행가가 있을 수 있다.

  Banker-schema = (지점명,고객명,은행가명)
  은행가명 → 지점명
  지점명 고객명 → 은행가명

  이 스키마에 BCNF 분해 알고리즘을 적용하면 다음과 같이 분해된다.

  Banker-branch-schema = (은행가명,지점명)
  Customer-banker-schema = (고객명,은행가명)

  이 분해에서 '지점명 고객명 → 은행가명'은 유지되지 않으며, 조인을 해야만 검사할 수 있다.

- R_1이 *Banker-branch-schema*이고 R_2가 *Customer-banker-schema*일 때, F의 제한 F_1과 F_2는 다음과 같다.

  $$F_1 = \{\text{은행가명} \rightarrow \text{지점명}\}$$
  $$F_2 = \emptyset$$

  따라서 $(F_1 \cup F_2)^+$에는 함수 종속 '지점명 고객명 → 은행가명'이 포함되지 않는다. 그러므로 $(F_1 \cup F_2)^+ \neq F^+$이기 때문에 이 분해는 종속 유지 분해가 아니다.

- 스키마는 여러 방법으로 BCNF 분해가 가능하다.
- 예2) 관계 스키마 $R(A, B, C)$에서 성립하는 함수 종속이 $A \rightarrow B$와 $B \rightarrow C$라 하자.

  – 함수 종속 $A \rightarrow C$를 이용하면 $R_1(A, B)$와 $R_2(A, C)$로 분해된다. 그러나 이 분해에서는 $B \rightarrow C$가 유지되지 않는다. 실제로 A는 수퍼키이므로 $A \rightarrow C$ 또는 $A \rightarrow B$는 분해할 때 이용되지 않는다.
  – 함수 종속 $B \rightarrow C$를 이용하면 $R_1(B, C)$와 $R_2(A, B)$로 분해된다. 그러나 이 분해에서는 $A \rightarrow B$, $B \rightarrow C$, $A \rightarrow C$가 모두 유지된다.

7.8 제3 정규형

- BCNF의 문제점은 종속이 유지되지 않을 수 있다는 것이다.
- 이것에 대한 해결책

  – 조인 계산을 하여 갱신의 유효성을 검사한다.
  – 3NF(Third Normal Form)을 사용한다. 이 정규형은 중복이 있을 수 있는 대신에 갱신 검사는 조인 없이 할 수 있다.

7.8.1 정의

정의 7.4 (3NF, Third Normal Form). 함수 종속 집합 F에 대해 관계 스키마 R이 다음을 만족하면 3NF에 속한다고 한다.
F^+에 있는 각 함수 종속 $\alpha \rightarrow \beta$은 다음 중 하나를 만족해야 한다. 여기서 $\alpha \subseteq R$이고, $\beta \subseteq R$이다.

- $\alpha \rightarrow \beta$는 자명한 함수 종속이다.
- α는 R의 수퍼키이다.
- $\beta - \alpha$에 있는 각 속성 A는 R의 후보키에 포함된다. □
- 3NF 정의의 마지막 조건에서 단일 후보키가 $\beta - \alpha$에 있는 모든 속성을 포함해야 하는 것은 아니다. 각 속성은 다른 후보키에 포함될 수 있다.
- BCNF에 속하는 모든 스키마는 3NF에 속한다.

F_c를 계산한다.
$i := 0$;
각 $\alpha \rightarrow \beta \in F_c$에 대해
  $\forall R_j$에 대해 $\alpha\beta \not\subseteq R_j$이면
    $i := i + 1$;
    $R_i := \alpha\beta$;
$\forall R_j$, R에 대해 후보키 $\not\subseteq R_j$이면
  $i := i + 1$;
  R_i := R의 아무 후보키;

<그림 7.9> 3NF 분해 알고리즘

- 예) BCNF을 설명할 때 사용한 은행가 스키마를 BCNF 분해를 하면 종속 유지가 되지 않는다. 그러나 은행가 스키마 자체는 3NF이다.
  - {지점명, 고객명}은 이 스키마에 후보키이므로 두 번째 함수 종속은 3NF을 위배하지 않는다.
  - 첫 번째 함수 종속에서 $\beta - \alpha$는 지점명이며, 지점명은 후보키 {지점명, 고객명}에 포함되므로 이 함수 종속도 3NF을 위배하지 않는다.
- 어떤 스키마가 3NF에 속하는지 검사하는 방법
  - F^+ 대신에 F만 고려한다.
  - F의 각 종속에 분해 규칙을 적용하여 오른쪽에 단일 속성만 있도록 한 다음에 고려한다.
  - 자명하지 않은 함수 속성 $\alpha \rightarrow \beta$가 주어지면 α^+를 이용하여 α가 수퍼키인지 검사한다.
  - 만약 α가 수퍼키가 아니면 $\beta - \alpha$의 모든 속성이 후보키에 포함되는지 검사한다. 이것은 어렵다.

7.8.2 분해 알고리즘

- 3NF 분해 알고리즘: 그림 7.9
- 예1) 은행가 스키마와 이 스키마에 성립하는 함수 종속이 다음과 같다고 하자.

  Banker-info-schema =
  (지점명,고객명,은행가명,사무실번호)
  은행가명 → 지점명 사무실번호
  지점명 고객명 → 은행가명

  이 예에서 $F_c = F$이다. 알고리즘을 적용하면 다음과 같이 분해된다.

  Banker-office-schema =
  (은행가명,지점명,사무실번호)
  Banker-schema = (고객명,지점명,은행가명)

- 3NF 분해는 F_c에 따라 또 F_c에 있는 종속을 어떤 순서로 고려했는지에 따라 달라질 수 있다.
- 알고리즘의 정확성: R_j가 이 알고리즘을 이용하여 R을 분해하였을 때 만들어진 스키마라 하자. R_j에서 성립하는 모든 자명하지 않는 함수 속성 $\gamma \rightarrow B$는 3NF을 충족해야 R_j는 3NF에 속한다. R_j를 생성할 때 사용된 함수 속성이 $\alpha \rightarrow \beta$라 하면 B는 α 또는 β에 포함되어야 한다.

  참고. 3NF을 검사할 때에는 함수 속성에서 오른쪽에 단일 속성이 있는 경우만 검사하는 것으로 충분하다.
  - B가 α와 β 모두에 있는 경우: F_c를 사용하므로 이것은 가능하지 않다.
  - $B \in \beta$이지만 $B \notin \alpha$인 경우
    - γ가 수퍼키인 경우: 3NF을 충족한다.
    - γ가 수퍼키가 아닌 경우: α에는 γ에 없는 속성을 가지고 있다. 그런데 $\gamma \rightarrow B \in F^+$이므로 γ^+를 이용하여 F_c로부터 $\gamma \rightarrow B$를 유도될 수 있어야 한다. 이 유도에서 $\alpha \rightarrow \beta$는 사용될 수 없다. 이것을 사용하기 위해서는 γ^+에 α가 포함되어야 한다. 하지만 γ가 수퍼키가 아니므로 이것은 가능하지 않다. 이 때 $\gamma \subseteq \alpha\beta$이고 $B \notin \gamma$(자명하지 않으므로)이므로 $\alpha \rightarrow (\beta - \{B\})$와 $\gamma \rightarrow B$로부터 $\alpha \rightarrow B$를 유도할 수 있다[1]. 이것은 B가 $\alpha \rightarrow \beta$에서 여분 속성임을 의미한다. 이것은 모순되므로 γ는 수퍼키이어야 한다.
  - $B \in \alpha$이지만 $B \notin \beta$인 경우: α가 후보키이므로 이것은 3NF의 세 번째 조건에 만족된다.
- 어떤 스키마가 3NF인지를 검사하는 것은 어렵지만 3NF 분해는 다항시간 내에 가능하다.

7.8.3 BCNF와 3NF의 비교

- BCNF와 3NF은 모두 무손실 조인 분해를 제공하지만 3NF은 함수 종속까지 유지해준다. 하지만 3NF은 BCNF와 달리 중복이 있을 수 있다. 또한 3NF의 경우에는 null 값을 사용해야 하는 경우도 발생한다.
  - *Banker-schema*의 경우에 담당 고객이 없는 경우에는 고객명에 null 값을 사용할 수밖에 없다.
  - 한 은행가가 여러 고객을 담당하면 은행가와 지점 관계가 여러 차례 중복되어 나타난다.
- SQL에서는 함수 종속을 지정하는 방법을 제공하지 않는다. 단정문을 사용할 수 있지만 함수 종속을 검사하는 단점문을 작성하기가 쉽지 않으며 비용이 비싸다. 따라서 함수 종속이 유지되는 분해를 사용하더라도 함수 종속의 왼쪽이 주키가 아니면 이것을 SQL에서 효율적으로 검사하기가 어렵다.

[1] $\delta = \beta - \{B\}$이면 $\alpha \rightarrow \delta, \alpha \rightarrow \alpha\delta, \alpha\delta \rightarrow \gamma$이므로 $\alpha \rightarrow B$이다.

- 함수 종속 $\alpha \rightarrow \beta$가 유지되지 않는 경우에는 이것을 검사하기 위해 분해를 조인한 다음 $\alpha\beta$만 추출하여 실체화 뷰를 만든 다음에 unique를 이용하여 이 함수 종속을 쉽게 검사할 수 있다.

- 일반적으로 3NF보다 BCNF를 사용하고 함수 종속이 유지되지 않는 경우에는 실체화 뷰를 이용하는 것을 선호한다.

7.9 제4 정규형

- 어떤 관계는 BCNF에 속하지만 여전히 중복 측면에서 충분히 정규화되지 않을 수 있다.

- 예) 다음과 같은 스키마를 생각해보자.

  BC-schema =
  (대출번호,고객명,고객-도시,고객-거리)

  이 스키마에서는 다음 함수 종속이 성립하므로 BCNF가 아니다.

  고객명 → 고객-도시 고객-거리

  그러나 만약 고객이 집을 여러 채 가지고 있으면 이 함수 종속이 성립하지 않으므로 BCNF에 속한다. 하지만 많은 중복을 가지게 된다.

- 이 처럼 BCNF에 속함에도 불구하고 중복이 있는 문제를 극복하기 위해 **다중값 종속(multi-valued dependency)**이라는 새로운 제약을 정의한다. 이 제약을 이용하는 정규형이 제4 정규형(4NF, Fourth Normal Form)이다.

7.9.1 다중값 종속

정의 7.5 (다중값 종속). 관계 스키마 R, $\alpha \subseteq R$, $\beta \subseteq R$이 있을 때, 다중값 종속 $\alpha \twoheadrightarrow \beta$가 R에 성립하기 위해서는 모든 적법한 관계 $r(R)$에서 $t_1[\alpha] = t_2[\alpha]$을 만족하는 r에 속한 모든 투플 쌍 t_1과 t_2에 대해 다음을 만족하는 투플 t_3과 t_4가 r에 존재해야 한다.

$$
\begin{aligned}
&t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\
&t_3[\beta] = t_1[\beta] \\
&t_3[R - \beta] = t_2[R - \beta] \\
&t_4[\beta] = t_2[\beta] \\
&t_4[R - \beta] = t_1[R - \beta]
\end{aligned}
$$

□

- 함수 종속은 특정 형태의 투플이 존재할 수 없도록 제약을 가한다. 예를 들어 $A \rightarrow B$가 성립하면 같은 A 값을 가지지만 서로 다른 B 값을 가지는 두 개의 투플이 존재할 수 없다.

- 다중값 종속은 특정 형태의 투플이 반드시 존재해야 하는 제약을 가한다. 이런 이유에서 함수 종속은 동일값 생성 종속(equality-generating tuple)이라 하고, 다중값 종속은 투플 생성 종속(tuple-generating tuple)이라 한다.

- 다중값 종속의 테이블 표현

	α	β	$R-\alpha-\beta$
t_1	$a_1, \ldots, a_i$	$a_{i+1}, \ldots, a_j$	$a_{j+1}, \ldots, a_n$
t_2	$a_1, \ldots, a_i$	$b_{i+1}, \ldots, b_j$	$b_{j+1}, \ldots, b_n$
t_3	$a_1, \ldots, a_i$	$a_{i+1}, \ldots, a_j$	$b_{j+1}, \ldots, b_n$
t_4	$a_1, \ldots, a_i$	$b_{i+1}, \ldots, b_j$	$a_{j+1}, \ldots, a_n$

- 다중값 속성은 α와 β 간에 관계가 α와 $R - \beta$ 간에 관계와 무관하다는 것을 말한다.

- $\beta \subseteq \alpha$이거나 $\beta \cup \alpha = R$이면 다중값 종속 $\alpha \twoheadrightarrow \beta$는 자명하다.

- 예) 다음과 같은 *bc*(*BC-schema*)를 생각하여 보자.

대출번호	고객명	고객-거리	고객-도시
L-23	유비	강남구	서울
L-23	유비	병천면	천안
L-27	유비	강남구	서울
L-27	유비	병천면	천안
L-93	장비	서초구	서울

  이 관계에서는 고객의 주소마다 고객의 대출번호를 중복시켜야 한다. 또한 고객의 각 주소마다 고객의 대출번호를 중복시켜야 한다. 따라서 이 관계에는 다음과 같은 다중값 종속이 성립한다.

  고객명 ↠ 고객-도시 고객-거리

  고객명 ↠ 대출번호

- D가 함수와 다중값 종속의 집합이면 D^+ 닫힘은 D에 의해 논리적으로 내포하는 모든 함수와 다중값 종속의 집합을 말한다.

- 다중값 종속의 정의에 의해 다음이 성립한다.
  만약 $\alpha \rightarrow \beta$가 성립하면 $\alpha \twoheadrightarrow \beta$가 성립한다.

7.9.2 4NF의 정의

정의 7.6 (4NF). 함수 종속과 다중값 종속 집합 D에 대해 관계 스키마 R이 다음을 만족하면 4NF에 속한다고 한다.
D^+에 있는 각 함수 종속 $\alpha \twoheadrightarrow \beta$은 다음 중 하나를 만족해야 한다. 여기서 $\alpha \subseteq R$이고, $\beta \subseteq R$이다.

- $\alpha \twoheadrightarrow \beta$는 자명한 다중값 종속이다.
- α는 R의 수퍼키이다.

□

- 4NF은 함수 종속 대신에 다중값 종속을 사용한 것을 제외하고는 BCNF와 같다.

- 모든 4NF은 BCNF에 속한다.

- R이 관계 스키마이고, R_1, R_2, ..., R_n이 R의 분해라 하자. 또한 D가 R에서 성립하는 함수 종속과 다중값 종속의 집합이라 하자. 그러면 R_i에 대한 D의 제한 D_i는 다음과 같다.
  - D^+에 있는 모든 함수 종속 중 R_i의 속성으로만 구성된 함수 종속
  - 다음과 같은 형태의 모든 다중값 종속

    $$\alpha \twoheadrightarrow \beta \cap R_i$$

    여기서 $\alpha \subseteq R_i$이고 $\alpha \twoheadrightarrow \beta \in D^+$이다.

result := $\{R\}$
done := false
D^+를 계산
done이 참이될 때까기 다음을 반복
  result에 D_i에 대해 4NF이 아닌 R_i가 있으면
    $\alpha \twoheadrightarrow \beta$가 다음을 만족하는 R_i에서
    성립하는 자명하지 않는 다중값 종속이면
      $\alpha \rightarrow R_i \notin D_i, \alpha \cap \beta = \emptyset$
    result := (result $- R_i$) $\cup$ ($R_i - \beta$) $\cup$ (α, β);
  없으면 done := true;

<그림 7.10> 4NF 분해 알고리즘

7.9.3 분해 알고리즘

- 4NF 분해 알고리즘: 그림 7.10
- 예) *BC-schema*에 4NF 분해 알고리즘을 적용하면 자명하지 않는 다중값 종속 '고객명 $\twoheadrightarrow$ 대출번호'를 발견하게 된다. 이것을 이용하여 분해하면 다음과 같다.

  Borrower-schema = (고객명,대출번호)
  Customer-schema =
  (고객명,고객-도시,고객-거리)

- 이 분해 알고리즘은 무손실 조인 분해를 제공하지만 종속 유지는 하지 못한다.

7.10 전체적인 데이터베이스 설계 과정

- 관계형 데이터베이스를 설계하는 세 가지 접근 방법
  - E-R 다이어그램을 설계한 후에 이것을 테이블로 변환
  - 필요한 모든 속성을 하나의 관계로 만든 후에 정규형 분해 알고리즘을 적용
  - 임의로 테이블을 만들고 정규형을 충족하는지 검사

7.10.1 E-R 모델과 정규화

- E-R 다이어그램을 올바르게 작성하고 이것을 테이블로 변환하면 더 이상의 정규화가 필요하지 않을 수 있다.
- 하지만 설계를 잘못하면 추가적인 정규화가 필요할 수 있다.
- 설계 과정에서 정규화 개념을 고려하여 설계하는 것이 필요하다. 이런 개념이 없는 경우에는 결과 테이블이 정규형을 충족하는지 검사할 필요가 있다.

7.10.2 범용 관계 접근 방법

- 필요한 모든 속성을 하나의 관계에 포함한 다음에 분해를 하기 위해서는 분해가 무손실 조인 분해이어야 한다.
- 이것이 가능하다는 것은 모든 관계를 조인한 하나의 관계가 유효해야 한다. 그러나 null 값의 사용하지 않고는 가능하지 않을 수 있다.
- 범용 관계(universal relation)란 데이터베이스에 있는 모든 관계를 조인하여 만든 관계를 말한다.
- 범용 관계의 모든 속성의 이름은 독특해야 한다.

7.10.3 성능을 위한 비정규화

- 가끔 데이터베이스는 고의로 정보가 중복된 스키마를 사용할 수 있다.
- 중복된 스키마를 사용할 때 문제점은 갱신할 때 많은 비용이 소요된다는 것이다. 이것을 희생하는 것이 오히려 비용 측면에서 더 효율적이면 중복이 있는 스키마를 사용할 수 있다.
- 그러나 비정규화된 데이터베이스를 사용하기 보다는 실체화 뷰를 사용하는 것이 바람직하다.

7.10.4 다른 설계 쟁점

- 정규화에 의해 해결되지 않는 문제점도 있다.
- 예) 회사의 수입에 관련한 다음과 같은 관계가 있다고 하자.

  earnings(회사명, 년도, 수입금액)

  이 관계에 존재하는 함수 종속은 '회사명 $\rightarrow$ 년도 수입금액' 밖에 없다. 따라서 이 관계는 BCNF에 속한다. 하지만 회사명이 중복된다. 다음과 같은 대안을 생각할 수 있다.

  - 대안 1. 해마다 새로운 관계를 생성한다.

    earnings-2003(회사명, 수입금액)

    이 관계 역시 BCNF에 속한다. 하지만 매 해마다 새로운 관계를 생성해야 하므로 좋은 해결책은 아니다.

  - 대안 2. 해마다 속성을 추가한다.

    company-year(회사명, 2001년-수입, 2002년-수입, 2003년-수입)

    이 관계 역시 BCNF에 속한다. 하지만 이 대안 역시 좋은 해결책은 아니다.