데이터베이스: 강의노트 03

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

Part I. Data Models

3 관계형 모델

3.1 관계형 데이터베이스의 구조

- 관계형 데이터베이스는 각각 독특한 이름을 가지는 테이블(table)의 모음으로 구성된다.

3.1.1 기본 구조

계좌번호	지점명	잔액
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

<그림 3.1> *account* 관계

- 테이블의 예: 그림 3.1 참조
- 테이블의 첫 행에는 각 열의 헤더를 나타낸다. 이 헤더들을 테이블의 속성(attribute)이라 한다.
- 각 속성이 가질 수 있는 값의 집합을 **도메인**(domain)이라 한다. 예) 지점명의 도메인: 은행의 모든 지점명
- 일반적으로 n개의 속성을 가지는 테이블의 i번째 속성의 도메인을 D_i라 하면 이 테이블은 다음 집합의 부분집합이다.

$$D_1 \times \cdots \times D_{n-1} \times D_n$$

- 수학에서 **관계**(**relation**)은 도메인 리스트의 카르데시안 곱(cartesian product)[1]의 부분집합으로 정의하고 있다. 관계형 데이터베이스의 테이블의 정의는 속성에 이름을 부여하는 것을 제외하고는 수학에서 관계의 정의와 같다.
- 앞으로는 테이블과 행 대신에 관계와 **투플**(**tuple**)이라는 수학 용어를 사용한다.

[1]수학에서 카르데시안 곱의 정의: 두 집합 A와 B의 카르데시안 곱은 가능한 모든 쌍 (a, b)의 집합을 말한다. 여기서 $a \in A$이고 $b \in B$이다.

- 투플 변수(tuple variable): 도메인이 모든 투플의 집합인 변수
  - *account* 관계는 7개의 투플로 구성되어 있다.
  - t를 첫 투플을 나타내는 변수라 하면, t의 특정 속성의 값을 나타내기 위해 다음과 같은 표기법을 사용한다.

    $$t[\text{계좌번호}] = \text{“A-101”}$$

    이 표기법 대신 보통 $t[1]$을 사용한다.
  - *account* 관계를 r로 나타내면 $t \in r$이라는 표기법을 사용하여 t가 관계 r에 속한 투플임을 나타낸다.
- 관계는 집합이므로 투플이 관계에 나타나는 위치는 아무런 의미가 없다.
- 원자적 도메인(atomic domain): 도메인의 원소를 더 이상 나눌 수 없는 도메인
  - 모든 관계의 속성의 도메인은 원자적이어야 한다.
  - 도메인 그 자체보다는 데이터베이스에서 그 도메인의 원소가 어떻게 사용되느냐가 더 중요하다.
- 여러 속성이 같은 도메인을 가질 수 있다. 논리적 관점(모든 고객의 이름)과 물리적 관점(문자열)에서 도메인을 접근할 수 있다.
- 모든 도메인에 속할 수 있는 유일한 값은 null 값이다.

3.1.2 데이터베이스 스키마

- 관계 스키마는 다음과 같이 나타낸다.

  $$\textit{Account-schema} = (\text{계좌번호}, \text{지점명}, \text{잔액})$$

  보다 정확한 스키마 정의는 속성 이름 옆에 그것의 도메인을 표기해야 한다.
- *account*가 *Account-schema*에 관한 관계임을 다음과 같이 나타낸다.

  $$\textit{account}(\textit{Account-schema})$$

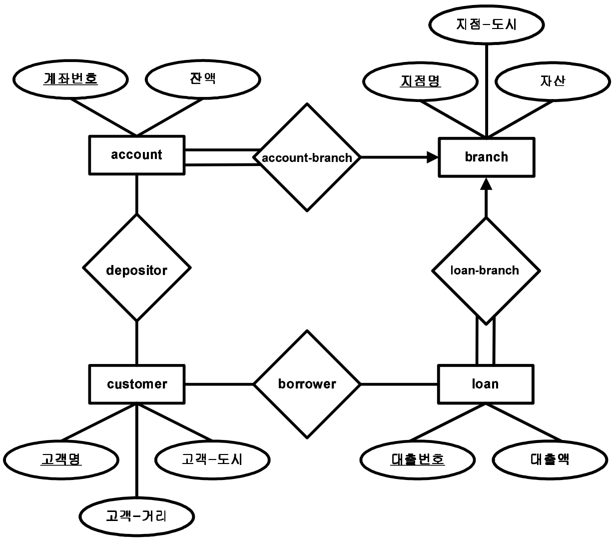
- *branch* 관계의 스키마가 다음과 같다고 하자.

  $$\textit{Branch-schema} = (\text{지점명}, \text{지점-도시}, \text{자산})$$

  branch 관계와 *account* 관계에는 둘 다 지점명이라는 속성을 가진다. 이런 중복은 서로 다른 관계의 투플 간에 관계를 나타내기 위해 사용된다.
- 은행 데이터베이스의 모든 스키마: 그림 3.2
  - 가정: 고객명은 독특하다. (원칙적으로는 각 고객에게 고유번호를 부여하여 이것을 주키로 사용해야 한다.)

Account-schema = (계좌번호, 지점명, 잔액)
Branch-schema = (지점명, 지점-도시, 자산)
Customer-schema = (고객명, 고객-거리, 고객-도시)
Depositor-schema = (고객명, 계좌번호)
Loan-schema = (대출번호, 지점명, 대출액)
Borrow-schema = (고객명, 대출번호)

<그림 3.2> 은행 데이터베이스의 스키마

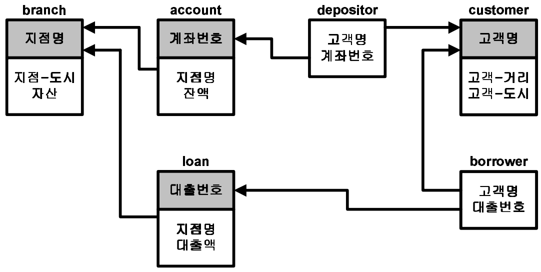


<그림 3.3> 은행 데이터베이스의 E-R 다이어그램

- 은행 데이터베이스의 E-R 다이어그램: 그림 3.3 참조.
  - *account-branch*와 *loan-branch*는 별도 테이블을 만들지 않고, *account*와 *loan* 관계에 포함하였다. 이것이 가능한 이유는 *account*에서 *branch*로, *loan*에서 *branch*로 다대일 관계이며, 각 관계에서 *account*와 *loan*은 전체적으로 참여하기 때문이다.

3.1.3 키

- E-R 모델에서 설명한 수퍼키, 후보키, 주키 개념은 관계형 모델에도 적용된다.
- R이 관계 스키마일 때 R의 부분집합 K가 R의 후보키가 되기 위해서는 $r(R)$의 모든 투플의 K의 값이 같지 않아야 한다. 즉, $t_1, t_2 \in r$이고 $t_1 \neq t_2$이면 $t_1[K] \neq t_2[K]$이어야 K가 r의 후보키가 될 수 있다.
- E-R 모델을 관계형 모델로 바꾸었을 경우 주키는 다음과 같이 결정된다.
  - 강한 개체 집합: 개체 집합의 주키가 관계의 주키가 된다.
  - 약한 개체 집합: 약한 개체 집합에 대한 관계는 약한 개체 집합의 속성과 그 집합의 식별 개체 집합의 주키를 속성으로 가진다. 이 때 관계의 주키는 식별 개체 집합의 주키와 약한 개체 집합의 부분키의 합집합이 된다.



<그림 3.4> 은행 데이터베이스의 스키마 다이어그램

  - 관계 집합: 관계 집합에 참여하는 개체 집합의 주키의 합집합이 관계의 수퍼키가 된다.
    - 관계의 대응수에 따른 주키: 2.3.2절 참조
    - 식별 관계는 별도의 관계를 만들지 않는다.
  - 테이블의 결합: A에서 B로 다대일 이진 관계 집합 R은 A의 속성과 관계 집합의 속성으로 구성된 하나의 관계로 개체 집합 A와 관계 집합 R을 모두 나타낼 수 있다. 이 때 개체 집합 A의 주키가 관계의 주키가 된다.
  - 다중값 속성: 다중값 속성은 개체 집합 또는 관계 집합의 주키와 다중값 속성을 나타내는 열로 관계를 만든다. 이 때 개체 집합 또는 관계 집합의 주키와 다중값 속성을 나타내는 새 열의 합집합이 관계의 주키가 된다.
- E-R 스키마에서 어떤 관계 r_1을 유도했을 때, 이 r_1은 다른 관계 r_2의 주키를 속성으로 가질 수 있다. 이 경우 이 속성을 r_2를 참조하는 r_1의 **외부키**(**foreign key**)라 한다. 또한 r_1을 참조하는 관계(referencing relation)라 하고, r_2를 참조된 관계(referenced relation)라 한다.
  - 예) *Account-schema*에서 "지점명"은 *Branch-schema*를 참조하는 외부키가 된다.
- 관계의 속성을 나열할 때 일반적으로 주키를 구성하는 속성을 가장 먼저 나열한다.

3.1.4 스키마 다이어그램

- **스키마 다이어그램**: 데이터베이스 스키마, 주키, 외부키의 의존성 등을 도식화할 때 사용하는 다이어그램
- 스키마 다이어그램의 구성요소
  - 각 관계는 박스로 나타내며, 박스 내부에 속성을 나열한다. 이 때 주키와 다른 속성은 가로선을 이용하여 분리한다.
  - 화살표를 이용하여 외부키의 의존성을 나타낸다.
- 스키마 다이어스램과 E-R 다이어그램의 차이점

– E-R 다이어그램은 외부키의 의존성을 명백하게 나타내지 않는다.
– 스키마 다이어그램은 관계의 대응수를 나타내지 않는다.

3.2 관계 대수

- **관계 대수(relational algebra)**는 절차식 질의어로 하나 또는 두 개의 관계를 입력으로 받아 그 결과로 새 관계를 출력해주는 연산으로 구성되어 있다.

3.2.1 기본 연산

3.2.1.1 선택 연산

- **선택(select)** 연산은 주어진 **조건 술어(predicate)**를 만족하는 투플을 찾아준다.
- 연산 기호: σ
- 예) Perryridge 지점에서 대출된 모든 대출을 찾아라.

$$\sigma_{\text{지점명}=\text{“Perryridge”}}(\textit{loan})$$

- 예) 대출액이 1200 이상인 모든 대출을 찾아라.

$$\sigma_{\text{대출액}>1200}(\textit{loan})$$

- 비교연산자: $=, \neq, <, \leq, >, \geq$
- 논리연산자: 논리곱($\wedge$), 논리합($\vee$), 부정($\neg$)
- 예) Perryridge 지점에서 대출된 대출 중에서 대출액 1200보다 큰 모든 대출을 찾아라.

$$\sigma_{\text{지점명}=\text{“Perryridge”}\wedge\text{대출액}>1200}(\textit{loan})$$

3.2.1.2 추출 연산

- **추출(project)** 연산은 관계을 축소하여 볼 수 있도록 해준다.
- 연산 기호: Π
- 예) *loan* 관계의 모든 투플의 대출번호와 대출액만 나열해라.

$$\Pi_{\text{대출번호},\text{대출액}}(\textit{loan})$$

3.2.1.3 관계형 연산의 혼합

- 연산의 결과는 그 자체가 관계이므로 연산 입력으로 관계 대신에 다른 연산의 결과를 사용할 수 있다.
- 예) Harrison 시에 거주하는 모든 고객의 이름을 찾아라.

$$\Pi_{\text{고객명}}(\sigma_{\text{고객-도시}=\text{“Harrison”}}(\textit{customer}))$$

3.2.1.4 합집합 연산

- **합집합(union)** 연산은 두 개의 연산 결과의 합집합을 구해준다.
- 예) 은행에 계좌 또는 대출을 가지고 있는 모든 고객의 이름을 찾아라.

$$\Pi_{\text{고객명}}(\textit{borrower}) \cup \Pi_{\text{고객명}}(\textit{depositor})$$

- 두 관계 r과 s에 합집합 연산을 적용하기 위해서는 다음 두 조건이 만족되어야 한다.
  – 조건 1. r과 s는 같은 수의 속성을 가져야 한다.
  – 조건 2. r의 i번째 속성의 도메인과 s의 i번째 속성의 도메인이 같아야 한다.

3.2.1.5 차집합 연산

- **차집합** 연산도 합집합 연산과 같은 조건을 만족할 때에만 적용할 수 있다.
- 예) 은행에 계좌만 있고 대출은 없는 모든 고객의 이름을 찾아라.

$$\Pi_{\text{고객명}}(\textit{borrower}) - \Pi_{\text{고객명}}(\textit{depositor})$$

3.2.1.6 카르데시안 곱 연산

- **카르데시안 곱** 연산은 어떤 두 개의 관계의 정보를 결합하는데 사용된다.
- 연산 기호: $\times$
- 두 개의 관계 r_1과 r_2는 같은 이름의 속성을 가질 수 있다. 이 두 관계의 카르데시안 곱을 구하기 위해서는 먼저 속성의 이름을 다시 정의해야 한다.
  – 예) $r = \textit{borrower} \times \textit{loan}$

$$(\textit{borrower}.\text{고객명}, \textit{borrower}.\text{대출번호}, \textit{loan}.\text{대출번호}, \textit{loan}.\text{지점명}, \textit{loan}.\text{대출액})$$

  – 보통 중복되지 않는 이름은 관계 이름을 생략한다.

$$r = (\text{고객명}, \textit{borrower}.\text{대출번호}, \textit{loan}.\text{대출번호}, \text{지점명}, \text{대출액})$$

  – 문제점
    - 같은 관계의 카르데시안 곱을 구할 경우
    - 연산의 결과와 다른 관계의 카르데시안 곱을 구할 경우
    - 해결책: 재명명(rename) 연산 사용
- r_1의 투플의 수가 n_1이고 r_2의 투플의 수가 n_2이면 $r = r_1 \times r_2$의 투플의 수는 $n_1 * n_2$이다.
- 예) Perryridge 지점에 대출이 있는 모든 고객의 이름을 찾아라.

– 단계 1. *borrower*와 *loan*의 카르데시안 곱을 구하고, 그 결과에서 지점명이 Perryridge인 투플을 찾는다.

$$\sigma_{\text{지점명}=\text{“Perryridge”}}(\textit{borrower} \times \textit{loan})$$

이 질의의 결과에는 *borrower*.대출번호와 *loan*.대출번호가 일치하지 않는 투플이 있을 수 있다. 이것은 의미가 없는 투플이다.

– 단계 2. 의미가 있는 투플만 선택한다.

$$\sigma_{\textit{borrower}.\text{대출번호}=\textit{loan}.\text{대출번호}} (\sigma_{\text{지점명}=\text{“Perryridge”}}(\textit{borrower} \times \textit{loan}))$$

– 단계 3. 위 질의의 결과에서 우리는 고객의 이름만을 원하므로 추출 연산을 추가로 적용한다.

$$\Pi_{\text{고객명}}(\sigma_{\textit{borrower}.\text{대출번호}=\textit{loan}.\text{대출번호}} (\sigma_{\text{지점명}=\text{“Perryridge”}}(\textit{borrower} \times \textit{loan})))$$

3.2.1.7 재명명 연산

- 연산의 결과는 관계이지만 이 관계는 이름이 없다. 연산의 결과에 이름을 할당하기 위해 사용하는 연산이 **재명명(rename)** 연산이다.
- 연산 기호: ρ
- 예) 관계 대수식 E의 결과를 x로 명명해라.

$$\rho_x(E)$$

- 예) 관계 대수식 E의 결과를 x로 명명하고, 그것의 각 속성의 이름을 A_i로 명명해라.

$$\rho_x(A_1, A_2, \ldots, A_n)(E)$$

- 예) 은행의 계좌 중에 잔액이 가장 많은 것을 찾아라.

  – 단계 1. *account* × *account*을 구한 후에 자기보다 큰 잔액을 가지고 있는 계좌를 모두 찾는다.

$$\Pi_{\textit{account}.\text{잔액}}(\sigma_{\textit{account}.\text{잔액}<d.\text{잔액}}(\textit{account} \times \rho_d(\textit{account})))$$

  – 단계 2. *account* 관계에서 잔액만을 추출한 것과 단계 1에서 얻은 결과의 차집합을 구하면 우리가 원하는 결과를 얻게 된다.

$$\Pi_{\text{잔액}}(\textit{account}) - \Pi_{\textit{account}.\text{잔액}}(\sigma_{\textit{account}.\text{잔액}<d.\text{잔액}}(\textit{account} \times \rho_d(\textit{account})))$$

3.2.2 부가 연산

3.2.3.1 교집합 연산

- **교집합** 연산도 합집합 연산을 적용하기 위해 만족해야 하는 조건을 충족해야 적용할 수 있다.
- 예) 은행에 계좌와 대출을 모두 가지고 있는 고객의 이름을 찾아라.

$$\Pi_{\text{고객명}}(\textit{borrower}) \cap \Pi_{\text{고객명}}(\textit{depositor})$$

- 교집합은 차집합 연산을 이용하여 나타낼 수 있다.

$$r \cap s = r - (r - s)$$

3.2.3.2 자연 조인 연산

- **자연 조인(natural join)** 연산은 카르데시안 곱과 선택 연산을 하나로 결합한 이항 연산이다.
- 연산 기호: $\bowtie$
- 두 관계 $r(R)$과 $s(S)$의 자연 조인 $r \bowtie s$는 스키마 $R \cup S$에 대한 관계이며, 그 정의는 다음과 같다.

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1=s.A_1 \wedge \cdots \wedge r.A_n=s.A_n}(r \times s))$$

  여기서 $R \cap S = \{A_1, \ldots, A_n\}$이다. 만약 $R \cap S = \emptyset$이면 $r \bowtie s = r \times s$이다.

- 예) 은행에서 대출을 받은 모든 고객의 이름과 그 고객의 대출액을 구하라.

$$\Pi_{\text{고객명},\text{대출액}}(\textit{borrower} \bowtie \textit{loan})$$

- 예) 은행에 계좌를 가지고 있고 Harrison 시에 거주하는 고객을 가진 모든 지점을 찾아라.

$$\Pi_{\text{지점명}}(\sigma_{\text{고객-도시}=\text{“Harrison”}}(\textit{customer} \bowtie \textit{account} \bowtie \textit{depositor}))$$

- 자연 조인 연산은 결합 법칙이 성립한다.

$$(r \bowtie s) \bowtie t = r \bowtie (s \bowtie t)$$

- 예) 은행에 계좌와 대출을 모두 가지고 있는 고객을 찾아라.

$$\Pi_{\text{고객명}}(\textit{borrower} \bowtie \textit{depositor})$$

  이것은 교집합 연산을 이용하여 찾을 수도 있다. 이 처럼 관계 대수에서는 같은 결과를 찾는 여러 형태의 대수식을 작성할 수 있다.

- 세타 조인(theta join): 자연 조인은 두 관계의 같은 속성을 기준으로 ‘=’ 조건 술어에 대한 선택 연산이 적용된다. 세타 조인은 이런 제한 없이 카르데시안 곱과 선택 연산을 하나의 연산으로 결합할 수 있도록 해준다. 세타 조인 $r \bowtie_\theta s$는 다음과 같이 정의된다.

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

3.2.3.3 나누기 연산

- **나누기(division)** 연산은 "모든 ∼에"이라는 절을 포함하는 질의어에 적합한 연산이다.
- 연산 기호: $\div$
- $S \subseteq R$인 두 관계 $r(R)$과 $s(S)$가 있을 때, $r \div s$는 $R - S$에 대한 관계이며, 어떤 투플 t가 $r \div s$에 속하기 위해서는 다음 두 조건을 만족해야 한다.
  - 조건 1. $t \in \Pi_{R-S}(r)$
  - 조건 2. $\forall t_s \in s$, $\exists t_r \in r$ s.t
    - a. $t_r[S] = t_s[S]$ and
    - b. $t_r[R - S] = t$
- 나누기 연산은 기본 연산을 이용하여 표현할 수 있다.

$$\begin{aligned} r \div s &= \Pi_{R-S}(r) - \\ &\quad \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)) \end{aligned}$$

- 예) Brooklyn에 위치해 있는 모든 지점에 계정을 가지고 있는 모든 고객의 이름을 찾아라.
  - 단계 1. Brooklyn 시에 위치해 있는 모든 지점

  $$r_1 = \Pi_{\text{지점명}}(\sigma_{\text{지점-도시="Brooklyn"}}(\mathit{branch}))$$

  - 단계 2. (고객명,지점명) 쌍

  $$r_2 = \Pi_{\text{고객명,지점명}}(\mathit{depositor} \bowtie \mathit{account})$$

  - 단계 3. $r_2 \div r_1$

$$\begin{aligned} &\Pi_{\text{고객명,지점명}}(\mathit{depositor} \bowtie \mathit{account}) \div \\ &\quad \Pi_{\text{지점명}}(\sigma_{\text{지점-도시="Brooklyn"}}(\mathit{branch})) \end{aligned}$$

3.2.3.4 배정 연산

- 관계 대수식의 일부분을 임시 **관계 변수**에 **배정(assignment)**하는 것이 편리한 경우가 있다.
- 연산 기호: $\leftarrow$
- 예) $r \div s$

$$\begin{aligned} \mathit{temp}1 &\leftarrow \Pi_{R-S}(r) \\ \mathit{temp}2 &\leftarrow \Pi_{R-S}((\mathit{temp}1 \times s) - \Pi_{R-S,S}(r)) \\ \mathit{result} &= \mathit{temp}1 - \mathit{temp}2 \end{aligned}$$

여기서 *temp*1과 *temp*2는 관계 변수이다.

3.3 확장 관계 대수

3.3.1 일반화 추출

- **일반화 추출(generalized projection)** 연산은 추출 리스트에 산술 함수를 사용할 수 있도록 일반 추출 연산을 확장한 연산이다.

고객명	신용한도	사용금액
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400

<그림 3.5> credit-info 관계

직원명	지점명	월급여
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Rao	Austin	1500
Sato	Austin	1600

<그림 3.6> pt-works 관계

- 일반화 추출 연산의 형식은 다음과 같다.

$$\Pi_{F_1,\ldots,F_n}(E)$$

  여기서 E는 관계 대수식이며, F_i는 E의 스키마 내의 속성과 상수로 구성된 산술식이다.
- 예) 그림 3.5에 기술된 *credit-info* 관계에서 각 고객의 남은 한도를 구하라.

$$\Pi_{\text{고객명,신용한도}-\text{사용금액}}(\mathit{credit\text{-}info})$$

- 일반화 추출 연산을 적용할 때 재명명 연산을 함께 적용할 수 있다.

$$\Pi_{\text{고객명,(신용한도}-\text{사용금액) as 남은한도}}(\mathit{credit\text{-}info})$$

3.3.2 집계 함수

- **집계 함수(aggregate function)**: 값의 집합을 입력으로 받아 하나 결과 값을 출력해주는 함수
- 대표적인 집계 함수: sum, avg, min, max, count
- 집계 함수는 값의 집합을 입력받지만 이 집합은 **다중 집합(multiset)**이다.
- 다중 집합: 중복되는 값이 존재할 수 있는 집합
- 집계 함수를 관계 대수식에 적용할 때 사용하는 형식은 다음과 같다.

$$_{G_1,\ldots,G_n}\mathcal{G}_{F_1(A_1),\ldots,F_m(A_m)}(E)$$

  여기서 G_i는 그룹핑의 기준이 되는 E의 속성이고, F_i는 집계 함수이며, A_i는 E의 속성이다.
- 예) 은행에 시간제로 근무하는 모든 직원의 월급여의 합을 구하라.

$$\mathcal{G}_{\text{sum(월급여)}}(\mathit{pt\text{-}works})$$

직원명	거리	도시
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

<그림 3.7> employee 관계

직원명	지점명	월급여
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

<그림 3.8> ft-works 관계

- 만약 집계 함수를 적용할 때 중복되는 값을 제거하고 싶으면 "-distinct"를 함수 이름 끝에 붙인다.
- 예) 시간제로 근무하는 직원이 있는 지점의 수를 구하라.

$$\mathcal{G}_{\text{count-distinct(지점명)}}(\textit{pt-works})$$

- 예) 각 지점별 시간제로 근무하는 모든 직원의 월급여의 합을 구하라.

$$_{\text{지점명}}\mathcal{G}_{\text{sum(월급여)}}(\textit{pt-works})$$

- 예) 각 지점별 시간제로 근무하는 모든 직원의 월급여의 합과 각 지점에서 가장 큰 월급여는 얼마인지 구하라.

$$_{\text{지점명}}\mathcal{G}_{\text{sum(월급여) as 월급여합, max(월급여) as 최대월급여}}(\textit{pt-works})$$

3.3.3 외부 조인

- 그림 3.7과 그림 3.8의 *employee*와 *ft-works* 관계의 자연 조인을 하면 Smith와 Gates에 관련된 정보는 얻지를 못한다. 이처럼 누락된 정보가 있을 때 사용하는 것이 **외부 조인(outer join)**이다.
- 외부 조인의 세 가지 형태
  - **좌측 외부 조인**(⟕): 자연 조인을 한 후에 좌측 관계에 조인하지 못한 모든 투플을 추가한다. 이 때 값이 없는 속성은 null 값으로 채운다.
  - **우측 외부 조인**(⟖): 자연 조인을 한 후에 우측 관계에 조인하지 못한 모든 투플을 추가한다. 이 때 값이 없는 속성은 null 값으로 채운다.
  - **완전 외부 조인**(⟗): 자연 조인을 한 후에 양쪽 관계에 조인하지 못한 모든 투플을 추가한다. 이 때 값이 없는 속성은 null 값으로 채운다.
- 외부 조인 연산은 기본 연산으로 나타낼 수 있다.

  $$\begin{aligned} & r ⟕ s = \\ & \quad (r \bowtie s) \cup \\ & \qquad ((r - \Pi_R(r \bowtie s)) \times \{(\text{null}, \ldots, \text{null})\}) \end{aligned}$$

  여기서 상수 관계 $\{(\text{null}, \ldots, \text{null})\}$의 스키마는 $S - R$이다.

3.3.4 Null 값

- null 값을 포함한 산술식의 결과는 null이다.
- null 값을 포함한 비교식의 결과는 unknown이다.
- unknown을 포함한 논리곱의 결과는 unknown이다.
- unknown과 참의 논리합은 참이며, unknown과 거짓의 논리합은 unknown이다.
- 각 연산에서 null 값 또는 unknown의 처리 방법
  - 선택: 조건 술어를 적용한 결과 값이 unknown 또는 거짓이면 결과에 포함되지 않는다.
  - 조인: 조인은 카르데시안 곱과 선택 연산을 하나로 합한 것이므로 선택 연산에 따른다.
  - 추출: 추출 연산은 중복을 제거할 때 null 값을 다른 값과 동일하게 취급한다. 만약 추출 결과 두 투플이 모든 필드의 값이 같으면 그 중에 null 값이 있던 없던 중복된 투플로 간주한다.
  - 합집합, 차집합, 교집합: 추출 연산과 같다.
  - 일반화 추출: 일반화 추출에서 산술식은 앞서 언급한 것과 같이 계산되며, 추출은 추출 연산과 같다.
  - 집계: 속성을 기준으로 그룹핑할 때 추출 연산과 같은 방법으로 null 값을 취급한다.

3.4 관계 대수 연산의 우선순위

- 우선순위 1. 단항연산자: σ, Π, ρ
- 우선순위 2. 곱셈연산자: $\times$, $\bowtie$, $\bowtie_\theta$
- 우선순위 3. 집합연산자: $\cup$, $\cap$, $-$, $\div$
- 이항연산의 경우 우선순위가 같으면 왼쪽에서 오른쪽으로 평가된다.
- 예) $R \cup \sigma S \bowtie T \equiv R \cup (\sigma(S) \bowtie T)$

3.5 데이터베이스의 수정

3.5.1 삭제

- **삭제(deletion)**를 관계 대수식으로 표현하면 다음과 같다.

  $$r \leftarrow r - E$$

  여기서 r은 관계이고 E는 관계 대수식이다.
- Smith의 모든 계좌 정보를 삭제해라.

  $$\textit{depositor} \leftarrow \textit{depositor} - \sigma_{\text{고객명="Smith"}}(depositor)$$

3.5.2 삽입

- **삽입(insertion)**을 관계 대수식으로 표현하면 다음과 같다.

$$r \leftarrow r \cup E$$

여기서 r은 관계이고 E는 관계 대수식이다.

- Smith가 Perryridge 지점에 계좌번호가 A-973이고 잔액이 1200인 계좌를 가지고 있다는 정보를 데이터베이스에 추가해라.

account ←
  account ∪ {("A-973","Perryridge",1200)}
depositor ←
  depositor ∪ {("Smith","A-973")}

3.5.3 갱신

- **갱신(update)**을 관계 대수식으로 표현하면 다음과 같다.

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(r)$$

여기서 r은 관계이며, r의 i번째 속성의 값이 변경되지 않으면 F_i는 속성을 나타내며, 변경되면 상수와 속성으로 구성된 산술식이다.

- 모든 계좌에 대해 5%의 이자를 지급해라.

$$account \leftarrow \Pi_{\text{계좌번호,지점명,잔액*1.05}}(account)$$

3.6 뷰

- 보안 문제 때문에 어떤 사용자에게는 데이터베이스의 일부만 접근할 수 있도록 해야 한다.
- **뷰(view)**: 사용자에게 제공된 데이터베이스의 논리 모델에 속하지 않는 가상 관계

3.6.1 뷰의 정의

- 뷰는 **create view** 문장을 이용하여 정의한다.
- **create view** 문장의 형식

create view v **as** ⟨query expression⟩

여기서 v는 뷰의 이름이다.

- 예) 지점과 고객명으로 구성된 뷰

create view *all-customer* **as**
  $\Pi_{\text{지점명,고객명}}$(*depositor* ⋈ *account*)∪
  $\Pi_{\text{지점명,고객명}}$(*borrower* ⋈ *loan*)

- **뷰와 배정 연산의 차이점**
  - 배정 연산을 이용하여 새롭게 만든 관계는 관련 관계의 데이터가 변경되어도 변하지 않는다.
  - 뷰는 그것을 사용할 때마다 새롭게 평가되어 사용되는 것이므로 뷰를 만들기 위해 사용된 관계의 데이터가 변경되면 뷰도 함께 변경된다.

- 뷰를 정의하면 데이터베이스는 보통 뷰를 계산하여 새로운 테이블을 만들지 않고, 그것의 정의만 저장한다. 나중에 그 뷰를 관계 대수식에서 사용하면 그 뷰의 이름 대신에 뷰를 정의한 관계식으로 대체되어 평가된다.
- 뷰를 정의하면 뷰를 계산하여 새로운 데이블을 만들어 저장하는 데이터베이스도 있다. 이런 뷰를 **실체화 뷰(materialized view)**라 하며, 이런 데이터베이스에서는 뷰가 현재 상태를 계속 반영하도록 계속 갱신해야 한다. 이런 작업을 뷰 유지보수(view maintenance)라 한다.

3.6.2 뷰를 통한 갱신

- 뷰를 통해 데이터베이스를 갱신하면 뷰와 관련된 실제 관계들도 갱신되도록 해야 한다. 이 때 뷰는 보통 기존 관계의 축소된 형태이므로 실제 관계를 갱신하기 위해 필요한 모든 정보가 없을 수 있다.
- 뷰를 통한 갱신의 두 가지 접근 방법
  - 방법 1. 무조건 거부
  - 방법 2. 실제 관계를 갱신하되 없는 정보는 null 값을 사용하여 채운다.
- 대부분의 데이터베이스는 방법 1을 사용한다.

3.6.3 다른 뷰를 이용하여 정의한 뷰

- 뷰 확장(view expansion): 한 뷰를 이용하여 새로운 뷰를 정의하는 것
- 뷰는 재귀적(recursively)으로 정의할 수 없다.

3.7 투플 관계 해석

- 관계 대수식은 절차식 언어이지만 투플 관계 해석(tuple relational calculus)은 비절차식 언어이다.
- 관계 대수는 연산 위주이며, 관계 해석은 정보 위주이다.
- 투플 관계 해석에서 질의의 형식은 다음과 같다.

$$\{t | P(t)\}$$

즉, 조건 술어 P를 만족하는 투플의 집합을 말한다.

3.7.1 질의의 예

- 예) 대출액이 1200 이상인 대출을 찾아라.

$$\{t \mid t \in loan \wedge t[\text{대출액}] > 1200\}$$

- 일부 속성만을 알고 싶은 경우에는 다음 표기법을 사용한다.

$$\exists t \in r(Q(t))$$

관계 r에 술어 조건 Q를 만족하는 어떤 투플이 존재한다.

- 예) 대출 중 대출액이 1200 이상인 대출의 대출 번호를 찾아라.

  $\{t \mid \exists s \in \textit{loan}(t[\text{대출번호}] = s[\text{대출번호}] \wedge$
  $s[\text{대출액}] > 1200)\}$

  t는 대출번호 속성에 대해서만 비교되므로 대출번호에만 국한된다.

- 예) Perryridge 지점에서 대출을 받은 모든 고객의 이름을 찾아라.

  $\{t \mid \exists s \in \textit{borrower}(t[\text{고객명}] = s[\text{고객명}] \wedge$
  $\exists u \in \textit{loan}(u[\text{대출번호}] = s[\text{대출번호}] \wedge$
  $u[\text{지점명}] = \text{“Perryridge”}))\}$

- 예) 은행에 계좌가 있거나 대출을 받은 모든 고객의 이름을 찾아라.

  $\{t \mid \exists s \in \textit{borrower}(t[\text{고객명}] = s[\text{고객명}]) \vee$
  $\exists u \in \textit{depositor}(t[\text{고객명}] = u[\text{고객명}])\}$

- 예) 은행에 계좌와 대출을 모두 가지고 있는 모든 고객의 이름을 찾아라.

  $\{t \mid \exists s \in \textit{borrower}(t[\text{고객명}] = s[\text{고객명}]) \wedge$
  $\exists u \in \textit{depositor}(t[\text{고객명}] = u[\text{고객명}])\}$

- 예) 은행에 계좌만을 가지고 있는 모든 고객의 이름을 찾아라.

  $\{t \mid \exists s \in \textit{depositor}(t[\text{고객명}] = s[\text{고객명}]) \wedge$
  $\neg\exists u \in \textit{borrower}(t[\text{고객명}] = u[\text{고객명}])\}$

3.7.2 수학적 정의

- 투플 관계 해석의 표현식의 형태는 다음과 같다.

  $$\{t | P(t)\}$$

  여기서 t는 투플 변수이고, P는 투플 관계식(tuple relational formula)이다.

- 투플 관계식 내에는 여러 투플 변수를 사용할 수 있다.

- 투플 변수가 $\forall$ 또는 $\exists$로 한정되지 않으면 자유 변수(free variable)라 하고, 한정된 변수는 한정 변수(bound variable)라 한다.

- 예) 다음에서 t는 자유 변수이고, s는 한정 변수이다.

  $t \in \textit{loan} \wedge \exists s \in \textit{customer}(t[\text{고객명}] = s[\text{고객명}])$

- 투플 관계 해석에서 투플 관계식은 원자(atom)로 구성되며, 그것의 형태는 다음과 같다.

  – $s \in r$: s는 투플이고, r은 관계이다. $\notin$ 연산자는 허용하지 않는다.

  – $s[x] \Theta u[y]$: s와 u는 투플이며, x와 y는 s와 u 각각에 정의된 속성이다. Θ는 비교 연산자($=, \neq, <, \leq, >, \geq$)이며, x와 y의 도메인은 이 속성에 의해 비교될 수 있어야 한다.

  – $s[x] \Theta c$: s는 투플이고, x는 s에 정의된 속성이다. c는 x의 도메인에 속한 상수이다.

- 이런 원자들을 다음 규칙에 적용하여 투플 관계식을 만든다.

  – 원자는 투플 관계식이다.

  – P_1이 투플 관계식이면 $\neg P_1$과 (P_1)도 투플 관계식이다.

  – P_1과 P_2가 투플 관계식이면 $P_1 \vee P_2$, $P_1 \wedge P_2$, $P_1 \Rightarrow P_2$도 투플 관계식이다.

  – $P_1(s)$가 자유 변수 s를 포함하는 투플 관계식이고 r이 관계이면 다음은 투플 관계식이다.

  $$\exists s \in r(P_1(s))$$
  $$\forall s \in r(P_1(s))$$

- 투플 관계식의 동등성

  – $P_1 \wedge P_2 \equiv \neg(\neg(P_1) \vee \neg(P_2))$

  – $\forall t \in r(P_1(t)) \equiv \neg\exists t \in r(\neg P_1(t))$

  – $P_1 \Rightarrow P_2 \equiv \neg(P_1) \vee P_2$

3.7.3 표현식의 안전성

- 다음과 같은 투플 관계 해석 표현식은 무한 관계를 생성할 수 있다.

  $$\{t \mid \neg(t \in \textit{loan})\}$$

  loan 관계에 속하지 않는 투플이란?

- 이 문제를 해결하기 위해 투플 관계식의 도메인이라는 개념을 사용한다. P의 도메인은 다음과 같이 표기하며,

  $$\textit{dom}(P)$$

  P가 참조하는 값의 집합을 나타낸다. 값의 집합에 포함되는 것은 P에서 참조하는 관계의 투플의 값과 P 자체에서 사용하는 값이다.

- 예) 다음 투플 관계식의 도메인은?

  $$\textit{dom}(t \in \textit{loan} \wedge t[\text{대출액}] > 1200)$$

  1200과 *loan* 관계의 등장하는 모든 값

- 예) 다음 투플 관계식의 도메인은?

  $$\textit{dom}(\neg(t \in \textit{loan}))$$

  loan 관계의 등장하는 모든 값

- 표현식의 결과에 등장하는 모든 값이 그 표현식의 도메인에 속하면 그 표현식은 안전하다고 한다. 따라서 $\{t \mid \neg(t \in \textit{loan})\}$는 안전하지 않다.

3.7.4 언어의 표현 능력

- 투플 관계 해석 언어는 관계 대수의 기본 연산으로 구성된 언어로 표현할 수 있는 모든 문장을 표현할 수 있다.