EE477 HW2

20170616 정희진

1. Relational Algebra

[Task]

(1) $\delta\left(\pi_{sin,firsName,lastName}\left(((\text{Employee} \bowtie \text{Customer}) \bowtie \text{Owns}) \bowtie \text{Account}\right)\right)$

(2) $\pi_{customerID}\left(\sigma_{branchName='Vancouver' \, OR \, branchName='Metrotown'}((\text{PersonalBanker} \bowtie \text{Employee}) \bowtie \text{Branch})\right)$

(3) $\delta\left(\pi_{sin,firstName,lastName,salary}\left((\text{PersonalBanker} \bowtie \text{Employee}) \bowtie_{sin=managerSIN} \text{Branch}\right)\right)$

(4) $\pi_{sin,salary}\left(\sigma_{salary>masterSalary}\left(\text{Employee} \bowtie \left(\pi_{salary \rightarrow masterSalary,branchNumber}(\text{Employee} \bowtie_{sin=managerSIN} Branch)\right)\right)\right)$

(5) $\delta\left(\pi_{customerID,firstName,lastName,income}\left(\sigma_{budget \leq 3200000}\left(((\text{Customer} \bowtie \text{Owns}) \bowtie \text{Account}) \bowtie \text{Branch}\right)\right)\right)$

(6) $\delta(\pi_{branchName}(\left(\pi_{branchNumber,branchName}(\sigma_{lastName='Martin'}(\text{Employee} \bowtie Branch))\right) \cap$
$\left(\pi_{branchNumber,branchName}\left(\sigma_{lastName='Jackson'}(\text{Employee} \bowtie Branch)\right)\right)))$

(7) $\delta\left(\pi_{customerID}\left(\text{Owns} \bowtie \left(\sigma_{numCustomer \geq 2}\left(\gamma_{accNumber,COUNT(customerID) \rightarrow numCustomer}(\text{Owns})\right)\right)\right)\right)$

(8) $\delta\left(\left(\pi_{firstName,lastName,birthData \rightarrow date}\left(\sigma_{branchName='London'}\left(((\text{Customer} \bowtie \text{Owns}) \bowtie \text{Account}) \bowtie \text{Branch}\right)\right)\right) \cup\right.$
$\left.\left(\pi_{firstName,lastName,startDate \rightarrow date}(\sigma_{branchName='London'}(\text{Employee} \bowtie \text{Branch}))\right)\right)$

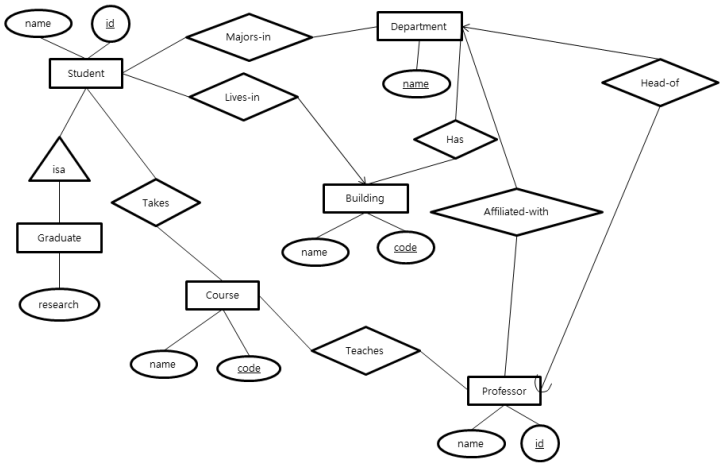
(9) $\pi_{firstName,lastName}(\sigma_{firstName='Steve' \, AND \, income<40000}(\text{Customer}))$

(10) $\left((\pi_{customerID}(\text{Customer})) - \left(\pi_{customerID}\left(\sigma_{amount<3000 \, AND \, amount>-3000}((\text{Customer} \bowtie \text{Owns}) \bowtie \text{Transactions})\right)\right)\right)$

2. Database Design

2.1 Entity-Relationship Model

[Task 1]



1. Student Entity set

포함해야 할 Student 정보 중 student ID와 name은 attribute로 설정하였다. Department와 Course는 넣어야 할 정보가 많기 때문에 따로 entity set으로 설정하였고 각각 Majors-in, Takes relationship으로 연결하였다. Student ID가 unique하기 때문에 primary key로 설정하였다. Dormitory는 Building으로 바꿔서 정보를 넣었는데, Building 또한 넣어야 할 정보가 많기 때문에 entity set으로 설정하였다. 이 둘은 Lives-in relationship으로 연결하였고 각각의 학생들은 최대한 한 dormitory안에 있기 때문에 Building entity set으로 향하는 normal arrow를 그렸다. Graduate student과 undergraduate student의 구분은 Graduate subclass를 만들면서 구분을 해놓았다. Graduate은 research attribute를 가지고 있다(undergraduate student 정보는 Student entity set에서 모두 구할 수 있고, graduate student만 subclass로 따로 빼주면서 research 정보를 추가적으로 포함시킬 수 있다. 이때 undergraduate student의 research 정보는 아예 없기 때문에 NULL을 포함시키지 않아도 되고, 그래서 [Task 2]에서 이 부분을 schema로 바꿀 때 E/R style conversion을 사용하였다).

2. Department Entity set

포함해야 할 Department 정보 중 department name은 attribute로 설정하였고 이것은 unique하기 때문에 primary key로 설정하였다. Professor과 Building은 넣어야 할 정보가 많기 때문에 따로 entity set으로 설정하였고 각각 Affiliated-with, Has relationship으로 연결하였다. 또한 각각의 부서는 단 한 명의 head professor가 있으므로 round arrow를 사용하여 Professor entity set과 Head-of relationship으로 연결하였다.

3. Professor Entity set

포함해야 할 Professor 정보 중 employeeID와 name은 attribute로 설정하였고 이중 employeeID는 unique하기 때문에 primary key로 설정하였다. Affiliated department는 위에서 설명한 것처럼 Department Entity set과 연결하여 정보를 얻었다. 또한 각각의 professor은 head가 아니거나 오직 하나의 부서를 맡은 head가 될 수 있으므로 normal arrow를 사용하였다. name of the courses the professor teaches 정보는 Course Entity set과 Teaches relationship을 사용하여 연결하고 얻

을 수 있게 하였다.

4. Course Entity set

포함해야 할 Course 정보 중 course name과 course code는 attribute로 설정하였고 이중 course code는 unique하기 때문에 primary key로 설정하였다. professors who teach it는 위에서 설명한 것처럼 Professor Entity set과 연결하여 정보를 얻었다.

5. Building Entity set

포함해야 할 Building 정보 중 building's name과 building code는 attribute로 설정하였고 이중 building code는 unique하기 때문에 primary key로 설정하였다.

[Task 2]

$$\text{Student}(\text{name}, \mathbf{id}, \text{dormitoryCode}^{FK-Building})$$

$$\text{MajorsIn}(\mathbf{studentID}^{FK-Student}, \mathbf{departmentName}^{FK-Department})$$

$$\text{Graduate}(\mathbf{id}^{FK-Student}, \text{research})$$

$$\text{Takes}(\mathbf{studentID}^{FK-Student}, \mathbf{courseCode}^{FK-Course})$$

$$\text{Course}(\text{name}, \mathbf{code})$$

$$\text{Building}(\text{name}, \mathbf{code})$$

$$\text{Teaches}(\mathbf{courseCode}^{FK-Course}, \mathbf{professorID}^{FK-Professor})$$

$$\text{Has}(\mathbf{departmentName}^{FK-Department}, \mathbf{buildingCode}^{FK-Building})$$

$$\text{Department}(\mathbf{name}, \text{headID}^{FK-Professor}) \text{ k: name}$$

$$\text{AffiliatedWith}(\mathbf{departmentName}^{FK-Department}, \mathbf{professorID}^{FK-Professor})$$

$$\text{Professor}(\text{name}, \mathbf{id})$$

대부분의 Entity set과 relationship은 relation으로 바꾸고 attribute도 각각의 attribute로 그대로 옮겼다(relationship은 연결된 entity set의 key를 attribute로 옮겼다). 이 때, Relationship에서 굵은 글씨는 primary key이고 FK라 명시된 것은 foreign key다. 특이한 것들만 몇 가지 설명하면 다음과 같다.

1. 너무 많은 relation은 좋기 않기 때문에 Student entity set과 Lives-in relationship을 하나의 relation으로 합쳤다. 따라서 Lives-in이 가지고 있어야 할 정보(Building-code)가 dormitoryCode로 이름이 바뀌어 attribute로 들어갔다.

또한 Department entity set와 Head-of relationship을 하나의 relation으로 합쳤다. 따라서 Head-of가 가지고 있어야 할 정보(Professor-id)가 headID로 이름이 바뀌어 attribute로 들어갔다.

2. Graduate subclass는 E/R style conversion을 이용하여 relation을 만들었고 이는 NULL value를 사용하지 않아도 된다는 장점이 있다. Object-oriented approach를 이용하여 Student을 Undergraduate로 바꾸어볼까 생각을 했지만 Student과 관계된 여러 relation이 많이 있고, Undergraduate와 Graduate를 하나의 Student로 보기 더 힘들다는 점이 있기 때문에 이렇게 바꾸었다.

2.2 Normal Forms

[Task 1]

FD1: User_ID → Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status

FD2: Product_ID → Product_Category_1, Product_Category_2, Product_Category_3

FD3: User_ID, Product_ID → Purchase

[Task 2]

다음 3개의 QUERY에서 모두 EMPTY SET으로 나오기 때문에 FD가 hold함을 볼 수 있다.

FD1:

```
SELECT User_ID, COUNT(DISTINCT Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status)
FROM Black
GROUP BY User_ID
HAVING NOT COUNT(DISTINCT Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status) = 1
AND NOT COUNT(DISTINCT Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status) = 0;
```

FD2:

```
SELECT Product_ID, COUNT(DISTINCT Product_Category_1, Product_Category_2, Product_Category_3)
FROM Black
GROUP BY Product_ID
HAVING NOT COUNT(DISTINCT Product_Category_1, Product_Category_2, Product_Category_3) = 1
AND NOT COUNT(DISTINCT Product_Category_1, Product_Category_2, Product_Category_3) = 0;
```

FD3:

```
SELECT User_ID, Product_ID, COUNT(Purchase)
FROM Black
GROUP BY User_ID, Product_ID
HAVING NOT COUNT(DISTINCT Purchase) = 1
AND NOT COUNT(DISTINCT Purchase) = 0;
```

[Task 3]

User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
1000001	P00069042	F	0-17	10	A	2	0	3			8370
1000001	P00248942	F	0-17	10	A	2	0	1	6	14	15200
1000001	P00087842	F	0-17	10	A	2	0	12			1422
1000001	P00085442	F	0-17	10	A	2	0	12	14		1057
1000002	P00285442	M	55+	16	C	4+	0	8			7969
1000003	P00193542	M	26-35	15	A	3	0	1	2		15227
1000004	P00184942	M	46-50	7	B	2	1	1	8	17	19215

1. Redundant information이 존재한다.

예를 들어 위 표에서 노란색으로 표시된 부분이 반복된 것을 볼 수 있다.

2. Update anomaly가 존재한다.

위의 표에서 1000001 User_ID를 가지고 있는 사람이 결혼을 해서 Marital_State가 1로 바뀌어야 할 때, 첫번째 tuple에서만 바뀐다면 나머지 tuple에서 잘못된 정보를 가지게 된다.

3. Deletion anomaly가 존재한다.

위 표에서 Product_ID가 P00285442인 상품을 없애야 할 때, 분홍색 부분이 삭제되어야 한다. 이때, User_ID가 1000002인 사람의 정보가 분홍색 tuple에만 있다면 그 유저의 정보를 더 이상 찾을 수 없을 것이다.

[Task 4]

BCNF decomposition 방법을 이용해 normalization을 해보았다.

Input:

FD1: User_ID → Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status

FD2: Product_ID → Product_Category_1, Product_Category_2, Product_Category_3

FD3: User_ID, Product_ID → Purchase

BlackFriday(USER_ID, Product_ID, Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2, Product_Category_3, Purchase)

Algorithm:

BlackFriday의 primary key는 User_ID, Product_ID이기 때문에 FD1은 BCNF에 위배된다(User_ID는 super key가 아니므로). 따라서 BlackFriday relation을

BlackFriday1(User_ID, Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status)와

BlackFriday2(User_ID, Product_ID, Product_Category_1, Product_Category_2, Product_Category_3, Purchase)로 나눈다.

이 때, BlackFriday1은 User_ID → Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status FD를 가지고 User_ID는 super key이므로 BCNF가 된다.

하지만, BlackFriday2는 Product_ID → Product_Category_1, Product_Category_2, Product_Category_3 FD에서 Product_ID가 super key가 아니기 때문에 BCNF에 위배된다. 따라서 BlackFriday2를

BlackFriday3(Product_ID, Product_Category_1, Product_Category_2, Product_Category_3)와

BlackFriday4(User_ID, Product_ID, Purchase)로 나눈다.

이 때, BlackFriday3은 Product_ID → Product_Category_1, Product_Category_2, Product_Category_3 FD를 가지고 Product_ID는 super key이므로 BCNF가 된다.

또한, BlackFriday4은 User_ID, Product_ID → Purchase FD를 가지고 User_ID, Product_ID는 super key이므로 BCNF가 된다.

따라서 BlackFriday relation은

BlackFriday1(User_ID, Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status)

BlackFriday3(Product_ID, Product_Category_1, Product_Category_2, Product_Category_3)

BlackFriday4(User_ID, Product_ID, Purchase)

로 decomposition 할 수 있다.

위의 relation들은 BCNF이기 때문에 3NF이면서 update anomaly, deletion anomaly가 존재하지 않는다. 또한 FD redundancy도 제거할 수 있다.

만약 BlackFriday relation의 MVD가 FD에서 확장된 MVD만 존재한다면

(Ex.User_ID ↠ Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status

Product_ID ↠ Product_Category_1, Product_Category_2, Product_Category_3

User_ID, Product_ID ↠ Purchase

또는 User_ID ↠ Gender처럼 한 개의 attribute가 오른쪽에 위치하는 MVD도 있음)

4NF decomposition을 해도 위와 똑같은 3개의 relation이 나올 것이다. 그렇다면 BlackFriday1, BlackFriday3, BlackFriday4는 4NF이기 때문에 MVD redundancy를 제거할 수 있을 것이다.