데이터베이스시스템 Project 1

20170364 김수빈

Index

1. Entity
   ① customer
   ② package
   ③ destination
   ④ timetable
   ⑤ history
   ⑥ contract
   ⑦ bill

2. Attribute
   ① customer
   ② package
   ③ destination
   ④ timetable
   ⑤ history
   ⑥ contract
   ⑦ bill

3. Relationship
   ① send
   ② shipment
   ③ reg_payment
   ④ tracking
   ⑤ itemize
   ⑥ ctmr_hist
   ⑦ ctmr_bill

4. Query Examples

5. Design Issues

1. Entity

① customer

고객 정보를 저장하기 위한 entity로, 쇼핑몰에서 회원 가입시 요구하는 기본적인 정보들을 담고 있다. 택배 회사에서 처리하는 업무의 가장 기본은 "**고객**이 보낸 택배를 배송지로 배송한다"이므로, customer 라는 entity가 필요하다.

② package

배송되는 물건에 대한 정보를 저장하기 위한 entity로, 명세서에 주어진 조건에 맞추어 물건에 대한 정보를 담는다. 택배 회사에서 처리하는 업무의 가장 기본은 "고객이 보낸 **택배**를 배송지로 배송한다"이므로, package 라는 entity가 필요하다.

③ destination

택배를 배송할 곳에 대한 정보, 수취인에 대한 정보를 저장하기 위한 entity로, 택배 회사에서 처리하는 업무의 가장 기본은 "고객이 보낸 택배를 **배송지**로 배송한다"이므로, destination 이라는 entity가 필요하다.
초기 구현 당시에는 customer 라는 entity 내에서 recursive relationship set을 정의하여 수취인과 배송지 정보를 사용하려고 하였으나, 택배를 보내는 사람은 택배 회사의 고객이지만 택배를 받는 사람은 택배 회사의 고객이 아닌 경우가 존재하기 때문에 destination 이라는 entity를 새로 생성하였다. 실제로 쇼핑몰에서 주문을 할 때, '배송지 저장' 기능이 있다는 것에서 착안하여 구현하였다.
반품의 경우, destination에서 출발하여 발송한 customer에게 돌아가는 형식인데, 이 경우에 대하여, 먼저 금액은 선불로 결제되므로 결제 금액을 저장할 필요는 없다. 따라서 이 경우에는 bill을 생성하지 않는다.

④ timetable

택배를 추적하기 위한 정보를 담기 위한 entity로, package에 의존하는 weak entity로 정의하였다. package 라는 entity 내에 해당 정보를 담지 않은 이유는, 택배가 여러 터미널 또는 집하지를 거쳐 최종 배송지까지 도착하기 때문에 package에 해당 정보를 담으면 계속 새롭게 update 되어 과거의 시점에 택배가 어디에 있었는지를 파악하는 것이 불가능하다. 따라서 새롭게 timetable 이라는 entity로 정보를 분리하여 정의하였다.

⑤ history

고객이 택배 발송 내역을 확인할 수 있도록 관련 정보를 저장하는 entity이다. customer entity에 의존하는 weak entity로 정의된다.

⑥ contract

월 정액제로 택배 회사와 계약을 맺은 회원들의 정보를 저장하기 위한 entity이다. customer entity에 의존하는 weak entity로 정의된다.

⑦ bill

고객들의 월별 총 결제금액을 저장하기 위한 entity이다. customer entity에 의존하는 weak

entity로 정의된다.

2. Attribute

*속성들 중 name, address, phone_num 등과 같이 이름만으로 자명한 속성의 경우 자세한 설명을 생략한다.

① customer

E-R model	Schema diagram
▪ customer_ID ▪ name ▪ address ▪ phone_num ▪ email	▪ customer_ID ▪ name ▪ address ▪ phone_num ▪ email
회원 정보의 구분은 중요하므로 확실하게 unique한 ID를 Primary key로 사용한다. 쇼핑몰 등 사이트에 회원 가입시 요구하는 기본적인 정보: 이름, 주소, 전화번호, 이메일을 저장한다.	ER model과 동일하다.

② package

E-R model	Schema diagram
▪ package_ID ▪ type ▪ weight ▪ timeliness ▪ caution_code	▪ package_ID ▪ customer_ID ▪ type ▪ weight ▪ timeliness ▪ caution_code ▪ address ▪ name ▪ phone_num

각 택배에 대한 unique한 ID를 생성하여 primary key로 사용한다. 명세서에서 주어진 조건에 의하여 속성을 생성하였다. - type: 택배의 종류를 나타낸다. flat envelope, small box, larger boxes - weight: 택배의 무게를 나타낸다. - timeliness: 배송 속도를 나타낸다. overnight, second day, loner - caution_code: 주의가 필요한 택배를 나타낸다. 유형별 코드는 임의로 설정한다.	customer와의 관계를 나타내는 send relationship에 대하여 customer 쪽이 one, package 쪽이 many 이면서 total participation이기 때문에 send relationship은 package에 customer의 primary key인 customer_ID 속성을 추가함으로써 표현할 수 있다. 마찬가지로 shipment 또한 동일하게 적용된다. 따라서 기존 속성에 customer의 primary key와 destination의 primary key가 추가된다.

③ destination

E-R model	Schema diagram
▪ address ▪ name ▪ phone_num	▪ address ▪ name ▪ phone_num
배송지의 경우, 한 주소에 여러 명이 살고 있을 수 있으므로 address 하나만은 primary key가 될 수 없다. 또한 기숙사와 같이 한 주소에 동명이인이 살고 있을 수 있으므로 address와 name으로는 primary key를 만족하지 않는다. 회사의 부서와 같이 전화번호를 공유하는 곳의 경우 동일한 전화번호를 사용하는 사람이 여러 명 존재할 수 있으므로 address와 phone_num으로도 primary key를 생성할 수 없다. 따라서 primary key는 모든 속성인 {address, name, phone_num}이 된다.	ER model과 동일하다. '1. Entity' 항목에서 설명했듯, '배송지 저장' 기능을 지원하고 있는 경우가 많기 때문에 현재 발송된 택배들의 배송지들의 집합과 저장된 배송지들의 집합이 항상 같지는 않다. 특정 고객에 의하여 저장은 되어 있는 배송지이지만 아직 그 곳으로 발송된 택배가 없는 경우가 존재하기 때문이다. 그렇기 때문에 destination 이라는 entity가 따로 존재한다.

④ timetable

E-R model	Schema diagram
▪ year ▪ month ▪ date ▪ time ▪ location ▪ transportation ▪ transp_id	▪ package_ID ▪ year ▪ month ▪ date ▪ time ▪ location ▪ transportation

▪ arrive	▪ transp_id ▪ arrive
weak entity이므로 자체적으로 primary key를 생성할 수 없다. - year, month, date, time: time stamp가 찍힌 시간을 나타낸다. - location: 지역을 나타낸다. - transportation: 어떠한 교통편으로 배송이 되었는지를 나타낸다. (ex. truck 1721의 "truck") - transp_id: 해당 교통편의 상세 정보를 나타낸다. (ex. truck 1721의 "1721") - arrive: 최종 배송지인지 여부를 나타낸다. 필수적인 attribute가 아니라 초기 구현에는 존재하지 않았으나, '배송지에 지정 시간 내에 도착했는지' 여부 등을 확인하기 위해서는 timetable entity의 tuple들과 destination entity의 tuple을 비교해야 하는 경우가 생긴다. string type인 address 속성의 특성상, 철자 하나 차이로 사실은 같은 주소인데 다른 주소로 인식되어(ex. "101-501"과 "101동 501호") 해당 결과를 찾지 못하는 문제가 발생할 수 있기 때문에 이를 막기 위하여 일종의 flag인 arrive 속성을 추가하였다. 코드는 임의로 설정 가능하다.(ex. 0: 미도착, 1: 도착)	package에 의존하는 weak entity이기 때문에 package의 primary key가 속성에 추가되었다. 이로써 tracking이라는 관계도 함께 정의되었기 때문에 tracking relationship은 별도의 relation으로 생성할 필요가 없다.

⑤ history

E-R model	Schema diagram
▪ year ▪ month ▪ date ▪ time ▪ price	▪ package_ID ▪ customer_ID ▪ year ▪ month ▪ date ▪ time ▪ price
weak entity이므로 자체적으로 primary key를 생성할 수 없다. - year, month, date, time: 택배 발송 시간을 나타낸다. - price: 택배 가격을 나타낸다.	package에 의존하는 weak entity이기 때문에 package의 primary key가 속성에 추가되었다. 이로써 itemize라는 관계도 함께 정의되었기 때문에 itemize relationship은 별도의 relation으로 생성할 필요가 없다. 또한 ctmr_hist relationship이 history 쪽에

	customer의 primary key를 추가함으로써 표현되었다. 따라서 ctmr_hist도 별도의 relation으로 생성할 필요가 없다.

⑥ contract

E-R model	Schema diagram
▪ duration ▪ pay_date ▪ pay_amount	▪ customer_ID ▪ duration ▪ pay_date ▪ pay_amount
weak entity이므로 자체적으로 primary key를 생성할 수 없다. - duration: 정액제를 이용할 기간을 나타낸다. 즉 계약 기간이다. - pay_date: 돈을 지불하게 되는 날짜를 나타낸다. 매 월 지불하게 되므로 날짜 정보만 저장하고 있는다. - pay_amount: 지불액을 나타낸다.	customer에 의존하는 weak entity이기 때문에 customer의 primary key가 속성에 추가되었다. 이로써 reg_payment라는 관계도 함께 정의되었기 때문에 reg_payment는 별도의 relation으로 생성할 필요가 없다.

⑦ bill

E-R model	Schema diagram
▪ year ▪ month ▪ owed	▪ customer_ID ▪ year ▪ month ▪ owed
- year, month: 청구 금액이 존재하는 년도와 월을 나타낸다. - owed: 해당 달에 결제해야 하는 총 금액을 나타낸다. 주의할 것은, contract entity의 tuple과 관계를 가지고 있는 customer의 경우는 어떤 history를 가지고 있느냐와 무관하게 정해진 금액이 매 달 owed로 저장된다는 것이다.	customer에 의존하는 weak entity이기 때문에 customer의 primary key가 속성에 추가되었다. 이로써 ctmr_bill이라는 관계도 함께 정의되었기 때문에 ctmr_bill는 별도의 relation으로 생성할 필요가 없다.

3. Relationship

① send

customer에서 package를 one-to-many로 설정한 이유는, 한 명의 고객이 여러 개의 택배를 발송할 수는 있지만, 하나의 택배가 여러 명의 고객에 의해 발송되는 경우는 없기 때문이다. 이 때, 택배를 발송하지 않은 고객은 있을 수 있지만 아무도 발송하지 않은 택배가 존재하지는 않기 때문에 package entity만이 total participation을 한다.

② shipment

package에서 destination을 many-to-one으로 설정한 이유는, 하나의 package는 오직 한 곳의 배송지를 갖지만, 한 곳의 배송지에는 여러 개의 택배가 도착할 수 있기 때문이다. 이 때 모든 택배는 배송지를 가져야 하기 때문에 package는 total participation을 하고, destination의 경우 db에는 저장되어 있는 배송지 목록이지만 아직 해당 주소로의 배송이 주문되지 않은 경우가 존재하기 때문에 destination은 partial participation이다.

③ reg_payment

regular payment, 즉 정기적으로 택배 시스템을 이용하는 고객을 나타낸다. one-to-one으로 설정한 이유는, 한 명의 고객은 하나의 계약을 가지게 되기 때문이다. 이 때 계약을 맺지 않은 infrequent customer가 존재하기 때문에 contract만이 total participation이다.

④ tracking

package에서 timetable을 one-to-many로 설정한 이유는, 하나의 package가 여러 단계에 걸쳐서 최종 배송지에 도착하기 때문에 여러 개의 timetable을 가지고 있을 수 있는 반면 어떠한 택배의 대한 추적 정보가 여러 개의 택배와 연결되어 있을 수 없기 때문이다. 또한 모든 package는 적어도 한 번 이상 추적되어야 하고, 반대로 모든 timetable의 tuple들은 그것이 추적하고 있는 택배가 있으므로 두 entity 모두 total participation이다.

⑤ itemize

one-to-one으로 설정한 이유는 하나의 package는 하나의 history로 남게 되고 하나의 history는 하나의 택배를 의미하기 때문이다. 따라서 itemize relationship은 one-to-one 관계이다. 또한 모든 history는 그것이 나타내고 있는 택배가 있고 반대로 모든 택배는 하나의 history가 되므로, 두 entity 모두 total participation이다.

⑥ ctmr_hist

customer에서 history를 one-to-many으로 설정한 이유는 한 명의 customer가 여러 개의 택배를 보내서 여러 개의 history를 가지고 있을 수 있지만 하나의 history가 여러 명의 고객에게 존재할 수는 없기 때문이다. 이 때, 모든 history는 어떠한 고객에 대한 history가 되지만 모든 고객이 history를 갖는 것은 아니므로, history entity는 total participation인 반면, customer는 partial participation이다.

⑦ ctmr_bill

bill에서 customer를 many-to-one으로 설정한 이유는 bill entity의 특성 때문으로, '이번 달에 한 명의 결제해야 할 금액' 만을 저장하고 있는 entity가 아니라, year, month 속성과 함께 해당 고객이 특정 기간에 얼마를 결제했는지를 전부 저장하고 있는 entity이기 때문이다. 따라서 한 명의 고객은 여러개의 bill을 가지게 된다. 이 때 모든 모든 청구액은 청구 대상 고객을 가지지만 모든 고객이 돈을 지불해야 하는 것은 아니므로 bill 측만이 total participation이다.

4. Query Examples

① Assume truck 1721 is destroyed in a crash. Find all customers who had a package on the truck at the time of the crash. Find all recipients who had a package on that truck at the time of the crash.

- truck 1721에 대한 정보는 timetable에서 얻을 수 있다. transportation = 'truck' and transp_id = 1721 인 튜플들을 찾은 후, 해당 튜플들의 package_ID에 접근하여 어떠한 택배가 사고가 발생한 차량에 실려있었는지를 알 수 있다. package_ID를 알고 나면 해당 택배를 발송한 고객이 누구인지, 그리고 해당 택배가 어디로 향하고 있었는지를 모두 구할 수 있다.

② Find the last successful delivery by that truck prior to the crash.

- timetable relation에서 arrive = 1인 튜플 중, 날짜 정보가 가장 최신인 건을 찾으면 된다.

③ Find the customer who has shipped the most packages in the past year.

- history relation에서 year 속성을 통해 작년에 발송된 택배들을 모두 찾은 후, 해당 택배 ID만을 추려낸다. 그런 후 history relation에서 customer_ID 중 가장 횟수가 많은 customer를 찾으면 된다.

④ Find the customer who has spent the most money on shipping in the past year.

- bill relation에서 year 속성을 통해 작년에 청구된 금액들을 나타내는 튜플들을 찾은 후, 같은 customer_ID에 대해 owed의 합을 구하여 총 합이 가장 큰 customer_ID를 찾으면 된다.

⑤ Find those packages that were not delivered within the promised time.

- timetable에서 각 package_ID별로 도착까지 걸린 총 시간을 계산한 후, 소요 시간이 해당 package_ID를 가진 package relation 에서의 튜플의 timeliness 정보와 비교하였을 때 더 큰 경우를 찾으면 된다.

⑥ Generate the bill for each customer for the past month. Consider creating several types of bills.

- A simple bill: customer, address, and amount owed.
- A bill listing charges by type of service.
- An itemize billing listing each individual shipment and the charges for it.

Customers like their bills to be readable. While the client will accept a relational table coming from Oracle as the bill, it would be "nice" to have a good-looking bill.

- simple bill의 경우, bill relation과 customer relation을 natural join하여 생성한다.
- bill listing charges by type of service의 경우, history relation과 package relation을 natural join 한 후, 각 customer_ID 에 대하여 type 속성별로 튜플을 구분한 뒤, 각 그룹 내에서 price 속성의 합을 구하여 생성한다.
- itemize billing의 경우, customer relation과 history relation을 natural join 하여 생성한다.

5. Design Issues

* 프로젝트 과정에서 고려했거나 전제한 것들에 대하여 주요 주제만 간단하게 서술한다.

① contract를 가진 고객과 infrequent 고객을 어떻게 구분하면 좋을지

- contract entity를 생성하여 해당 entity 내에 customer_ID 속성에 대하여 존재하는 고객의 경우는 frequent customer라고 구분할 수 있다.

② billed monthly 는 어떻게 체크해야 하는지

- 초기 구현 당시에는 bill entity에 owed만이 존재하였으나, 다달이 금액이 후불로 청구되는 것을 고려하였을 때 고객이 과거 청구 내역을 조회할 수 있고 또 월마다 청구될 금액이 초기화되어야 하기 때문에 year과 month 속성을 bill entity에 추가하여 이를 구분하였다.

③ 무조건 회원이어야 택배를 보낼 수 있는지

- 보내는 경우에는 그렇다. (반품의 경우, 회원이 아니라면 '비회원 발송' 등으로 처리 가능. 이 경우에는 bill을 생성하지 않는다.) 그러나 받는 사람의 경우에는 회원이 아닐 수도 있기 때문에 destination 이라는 entity 를 추가로 생성한다.

④ account number와 credit card 정보를 저장하지 않는 이유

- contract를 맺은 정기 고객의 경우, 월마다 정액제로 account number를 통하여 돈을 지불한다고 하였다. 이 때의 account number는 택배 회사의 account number라고 해석하였고, 즉 계약을 맺은 고객들의 경우 계좌 이체를 통하여 돈을 지불하면 된다는 것으로 이해하였다. 따라서 '계좌 번호' 자체는 중요하지 않고 내야 하는 금액이 중요하므로 account number는 따로 저장하지 않는다. credit card 정보 또한, 고객이 후불로 결제해야 하는 금액 정보가 중요한 것이고 카드 개인정보 자체는 택배회사 측에서 알지 않아도 되기 때문에 저장하지 않았다.

⑤ contract는 택배 회사와 고객 사이에 하나뿐인지

- 이는 design issue라고 생각하여, 한 명의 고객이 두 개의 계약을 가질 수는 없다고 전제하고 진행하였다. 기존에 계약을 맺고 있던 고객이 계약 조건(duration, pay_date 등)을 바꿀 수 있지만, 이러한 경우에는 새로운 tuple을 생성하는 것이 아니라 해당 고객과 연결되어 있던 기존의 contract tuple을 update하는 방식으로 진행한다.