과목명: 데이터베이스시스템

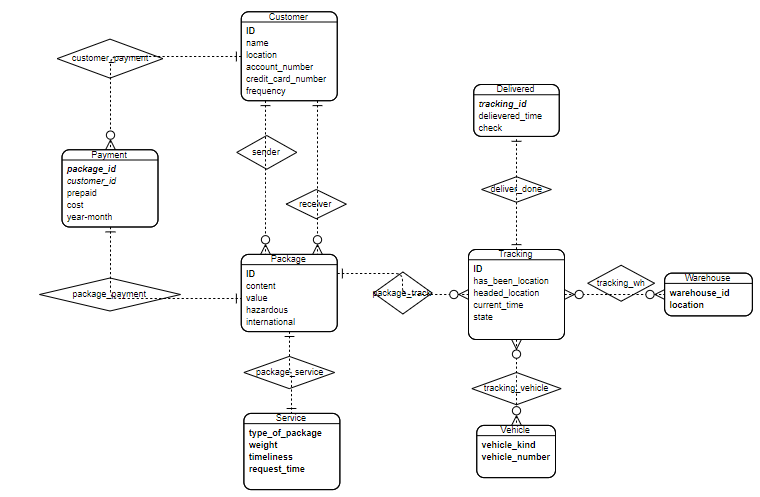
<project #1>

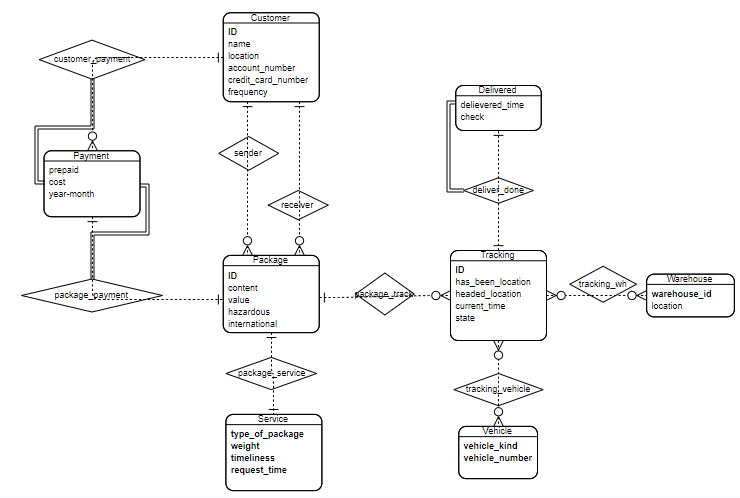
서강대학교 수학과

20141362

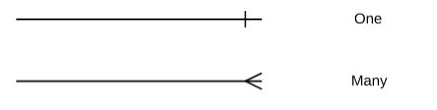
천지환

1. **ER diagram**



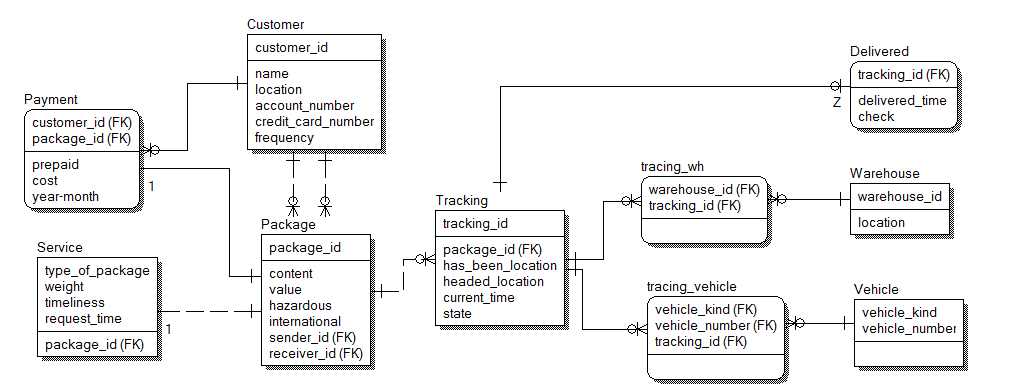


-명세서에 주어진 tool이 weak entity를 표시하기 어려워 첫번쨰 diagram은 weak entity여도 모든 attribute를 쓴 상태이고 두번째 diagram은 weak entity 형식에 맞춰 attribute를 제거하고 weak entity의 경우 entity와 relation을 추가로 double line으로 이어준 상태입니다. Cardinality notation은 아래 그림과 같은 notation을 이용하였습니다.



원래 many 앞에 0표시가 있을 경우 zero or many를 나타내지만 tool에서 zero or many와 many를 구분하는 line과 zero or one과 one을 구분하는 line이 없었기에 이는 구분하지 않았습니다.

1. **Relation schema diagram**

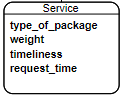


1. **ER diagram 구성**

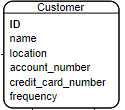
명세서에 따라서 ER diagram의 entity와 relation 그리고 그 attributes를 결정 했습니다. 먼저 프로젝트 명세서의 가장 처음에 나와있듯 이 프로젝트는 배송 회사에서 요청한 프로젝트로 제목 및 큰 개요를 package delivery로 잡았습니다.

첫번째 조건에서 type of package, weight of the package, timeliness of delivery에 따라 다른 종류의 service가 주어진다 하였기에 service entity를 만들고 그 attribute으로 type\_of\_package, weight, timeliness로 잡았습니다. ER diagram에는 표시하지 않았지만 type\_of\_package는 첫번째 조건에 따라 (flat envelope, small box, larger boxes) 중 하나의 값을 가지고 timeliness는 (overnight, second day, longer)라는 정보에서 longer을 조금 더 구체화 하여 배송에 걸리는 일수를 갖는 값으로 설정했습니다. 그리고 뒤에 나오는 query문에서 예상 배송시간보다 더 오래 걸린 package 들을 구하는 항목이 있는데 이때에 사용하기 위해서 주문 요청을 한 시간인 request\_time을 attribute으로 추가하였습니다. 이 attribute는 YYYY-MM-HH의 값을 갖도록 설정하였습니다.(request\_time과 timeliness를 이용하여 예상 도착 일자를 알 수 있습니다.)

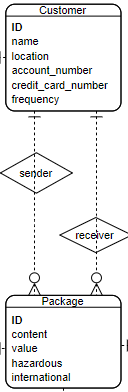
그 다음 이 4개의 항목을 모두 엮어서 primary key로 설정하였습니다 type\_of\_package, weight, timeliness, request\_time 까지는 겹칠 수 있지만 4개 항목이 request\_time까지 겹치는 경우는 없다고 생각했습니다.



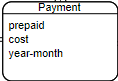
두번째 조건에서는 고객들의 지불 방법에 대해 미리 회사와 계약한 고객님은 달마다 정기적으로 계좌 번호를 통해 지불하고 그렇지 않은 고객은 구매할 때마다 그들의 신용카드를 이용하며 반송 제품에 대해선 prepaid 되어있다고 하였습니다. 따라서 먼저 sender와 receiver에 대한 entity를 만들려고 고려했으나 둘 모두 필요한 attribute들의 값이 똑같다고 생각해 customer이라는 attribute를 먼저 만들었습니다. Customer의 attribute로는 customer의 id, name, location(사는 지역), account\_number, credit\_card\_number. Frequency 값들을 지정했습니다. Name과 location은 배송회사가 고객에게 배송하기 위해 당연히 가지고 있어야 할 정보이며 비정기 고객이 정기고객으로 전환 될 수도, 정기 고객이 비정기 고객으로 전환 하거나 반송 물품이 있어서 prepaid가 필요할 상황이 있을 수도 있기에 account\_number와 credit\_card\_number 모두를 attribute 값으로 설정했습니다. 그리고 비정기 고객과 정기 고객의 구분을 위해 frequency를 attribute에 추가 해주었습니다. ERdiagram에는 나와 있지 않지만 이때 freqency의 값은 yes or no의 값을 갖도록 설정 하였습니다. 그리고 customer의 id를 primary key로 설정하였습니다.



그 후 sender와 receiver가 구분되지 않기에 package entity를 만들어 일단 attribute로 id를 넣어준 다음 customer와의 relation으로 sender와 receiver 2개를 만들어 주었습니다. 이때 customer는 package를 여러 개 보내고 받을 수 있기에 customer와 package의 relation은 one-to-many의 관계를 설정하였습니다.

(그림에서는 후에 package에 후에 추가된 attribute까지 존재)

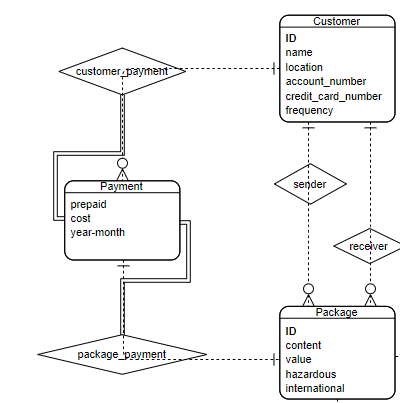
그 후 bill에 대한 정보를 알기 위해 payment라는 entity를 만들어 반송 되어서 미리 지불된 것인지 알기 위해 prepaid를, 배송 비용을 알기 위해 cost를, 정기 고객이 달 단위로 계산하거나 고객의 년도에 따른 비용을 알기 위해 year-month를 attributes로 설정하였습니다.



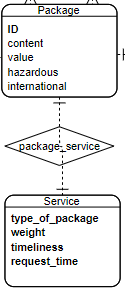
이때 payment는 customer와 package와 package에 대한 정보이기에 cutomer와 customer\_payment라는 이름으로 relation을 연결 해주었고 customer 한명이 여러 package에 대한 bill을 가질 수 있기에 customer와 payment의 관계는 one-to-many로 설정하였습니다. 그리고 payment에서 customer의 id를 foreign key로 가져와 primary key의 요소 중 하나로 삼게 설정하여 weak entity로 만들었습니다.

Customer 뿐만 아니라 package와도 연관 있기에 package\_payment라는 relation을 만들고 package 하나당 하나의 bill 정보가 필요하기에 one-to-one 으로 설정하였고 package\_id를 foreign키로 받아와 primary key의 요소 중 하나로 삼게 설정하여 weak entity로 만들었습니다.

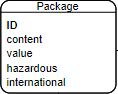
이렇게 해서 payment는 (customer\_id, package\_id)를 primary key로 설정하였니다.



생겨난 entity들 중 기존에 있던 service entity와 관련이 있는건 package entity라고 생각해 둘 사이를 package\_service relation으로 만들고 package 하나당 service하나의 정보가 연결되기에 one-to-one으로 구성하였습니다.



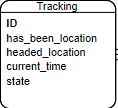
세번째 조건에서 배송 회사는 hazardous material이 포함되어 있을 시 내용물 확인이 필요하고 해외 배송에서 관세 부과시 내용물과 값어치 비교가 필요하다고 하였기에 두번째 조건에서 만들었던 package entity에 위험 물질인지 체크하는 hazardous attributes와 해외 배송인지 확인하는 international attribue를 추가 하고 이때 필요한 content와 value attribute를 추가 하였습니다. Package의 primary key로는 기존에 있던 id 값으로 설정하였습니다.



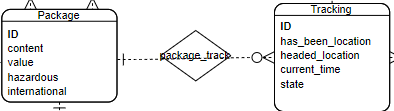
네번째 조건에서 company와 customer들은 package를 보낼 때부터 받을 때까지 계속 tracking해야되며 이때 어디에 있는지, 어디에 있었는지, 현재 어디로 향하는지를 알 수 있어야 하고 운송 수단이나 보관 warehouse까지 알 수있어야 한다고 하였습니다. 따라서 저는 tracking entity를 만든 후에 이를 package 하나 당 여러 개의 값을 갖게 하여 package 하나의 배송 정보를 계속하여 갱신하고 전의 정보들 또한 갖을 수 있도록 설정하였습니다.

Tracking의 attribute로는 일단 primary key 값을 갖는 id를 설정하였고, 그후 조건의 정보에 따라 has\_been\_location, headed\_location을 추가 하였습니다. 그리고 추가로 current\_time항목을 넣었습니다. 이는 treacking 값이 has\_been location과 headed\_location이 바뀔때마다 새로 instance를 만들어 package에 추가 하는 형태로 구성했는 데 이때 도착 시간을 알기 위해서 current\_time을 설정 하였으며 instance를 위치가 바뀔 때 뿐만 아니라 30분 주기로 tracking을 최신화 해주기 위해서도 이용됩니다.

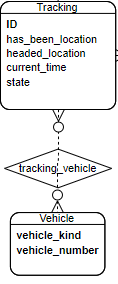
뿐만 아니라 운송 수단에 대한 정보도 이용하기 위해 tracking에 state attribute를 만든 다음 state값은 배송 준비, 배송 시작, 이동 중, 물류 창고, 도착 완료의 값들을 가지게 설정하였습니다. Has\_Been\_location과 headed\_location 뿐만 아니라 이 state 값이 바뀔 때 마다도 tracking을 업데이트 하게 설정하였습니다.



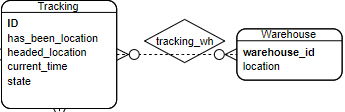
이때 id를 primary key로 잡아 주었으며 기존 entity들 중에는 package와 관련이 있고 package 하나당 여러 개의 tracking 정보를 갖을 예정이기에 package\_track이라는 이름의 one-to-many relation을 만들어 주었습니다.



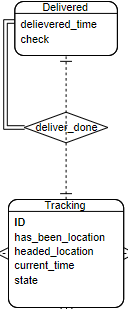
그리고 state에서의 이동 중, 물류 창고, 배송 완료에 대한 자세한 정보를 주기 위해 추가로 entity들을 만들었습니다. 먼저 vehicle entity를 만들어 이동 중에 어떤 종류의 운송 수단을 이용하는 지 vehicle\_kind attribute와 운송 수단의 차량 번호(비행기 번호나 선박 번호도 포함) vehicle\_number attribute를 만들어 넣어 주었습니다. 그리고 (vehicle\_kind, vehicle\_number)을 primary key로 설정 하였고 tracking 하나 당 여러 개의 vehicle을 가질 수 있고 또 여러 tracking이 같은 vehicle을 사용할 수 있기에 tracking\_vehicle이라는 이름의 many-to-many relation을 설정하였습니다.(기종이 다를 경우 운송 수단 번호가 같은 수도 있다고 생각하기에 둘을 묶어서 primary key로 설정)



물류 창고에 대한 정보는 warehouse entity를 만들어 warehouse\_id와 location을 entity로 넣어 주었으며 이도 운소수단과 마찬가지고 tracking 하나 당 여러 개의 warehouse를 가질 수 있고 또 warehouse 하나 당 여러 개의 tracking을 갖을 수 있기에 tracking\_wh라는 이름의 many-to-many relation을 만들어 주었습니다.



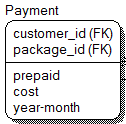
그리고 물품 배송이 완료된 항목들에 대해서 고객이 해당 물품을 받은 것을 체크하기위해 Delivered entity를 만들고 도착 시간을 저장하는 delivered\_time attribute을 만들고 고객이 받았는지 여부를 확인하는 check attribute을 추가하였습니다. 이 delviered는 tracking의 primary key인 tarcking\_id를 foreign key로 받아와서 primary key로 이용하기에 weak entity이고 tracking의 state가 배송 완료가 아닐 경우 delivered entity와 relation이 없고 배송 완료일 경우 1:1로만 relation이 생기기에 zero,one-to-one 관계로 설정하였습니다. Relation의 이름은 deliver\_done으로 설정하였습니다.



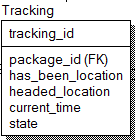
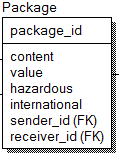
1. **Relation schema diagram 구성**

교재에 나와있는 ERdiagram을 relation schema로 reduction하는 과정에 맞게 erdiagram을 reduction하였습니다.(many-to-many의 경우를 제외하고 cardinarlity는 ERdiagram과 동일)

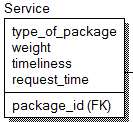
먼저 weak entity의 attribute들에 identyfing entity에서 가져온 primary key 값들을 추가 해서 foreign key이자 primary key로 설정 해준 다음에 relation을 생략 했습니다. 따라서 payment attribute에 customer\_id 와 package\_id attribute이 추가되고 customer\_payment, package\_payment relation이 생략되었습니다. 또한 delivered entity에 tracking\_id 를 foreign key이자 primary key로 추가 해주고 deliver\_done relation은 생략해 주었습니다.



그리고 many-to-one relation의 경우 relation을 생략해주고 many쪽에 상대 primary key를 attribute를 추가하면 되기에 customer package의 relation인 sender와 receiver을 생략하고 package에 sender\_id와 receiver\_id를 foreign key로 추가 해 주었습니다. 그리고 package, tracking의 relation인 package\_track을 생략하고 tracking에 package\_id를 foreign key로 추가해주었습니다.

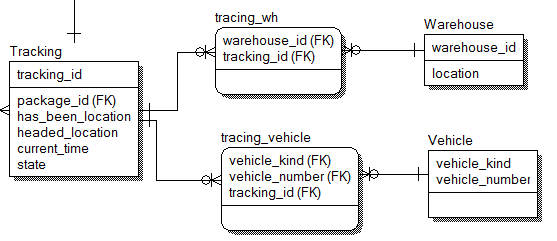


One-to-one relation의 경우 relation을 생략하고 둘중 아무 entity쪽에나 상대 primary key를 attribute를 추가해주면 되기에 package,service의 relation인 package\_service relation을 생략하고 service에 package\_id를 foreign key로 추가해 주었습니다.



Many-to-many relation의 경우 relation entity를 새로 만들고 두 entity의 primary key를 foreign key로 가져와 primary key로 설정해야 하기에 tracking\_wh entity를 새로 만들어서 warehouse의 primary key인 warehouse\_id와 tracking의 primary key인 tracking\_id를 foreign key로 가져와 primary key로 설정해 주었습니다. 기존 entity와 relation entity의 cardinarlity는 one-to-many로 설정 하였습니다.(many-to-many를 relation entity로 만들어 분할 하였기에 entity-relation entity-entity에서의 관계는 one-to-many/ many-to-one)

Tracing\_vehicle relation도 마찬가지로 새로운 entity로 만들어 tracking의 primary key인 tracking\_id와 vehicle의 primary key인 (vehicle\_kind,vehicle\_number)를 foreign key로 가져와 primary key로 설정하였습니다. Relation은 tracing\_wh와 마찬가지로 relation entity와 기존 entity를 각각 many-to-one으로 설정하였습니다. (many-to-many를 relation entity로 만들어 분할 하였기에 entity-relation entity-entity에서의 관계는 one-to-many/ many-to-one)



1. **Query문(첫번째 문제에 대해 먼저 풀었는데 설명이 너무 복잡하고 이해하기 힘들 것 같아서 2개 풀었습니다! 첫번째 것은 안보셔도 됩니다!)**

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

먼저 truck이 해당 사고 시간에 가지고 있었던 모든 물품 들을 확인하기 위해서 tracking entity를 먼저 package\_id로 grouping 한다. 그 후 group current\_time 값이 max 값인 instance 들을 select한다. 그 중 state가 (이동 중) 인 instance들을 다시 select 한다. 그 뒤 이 instance들의 tracking\_id와 vehicle entity의 tracking\_id를 이용해 natural join을 시행한다. 그러면 vehicle의 vehicle\_kind와 vehicle\_number을 통해 사고난 truck을 특정할 수 있고 이때 운송중이던 tracking\_id 또한 알 수 있다. 이 tracking\_id 들에 해당하는 package\_id 들을 이용해 crash 된 package들을 구한다.

(이때 tracking은 특정 시간마다, 운송 수단 및 배송 목적지가 바뀔 때 마다 최신화 되므로 각 group의 current\_time의 값이 max 값을 갖는 instance는 해당 package의 최신 위치이다.)

이 트럭이 사고 나기전 가장 마지막으로 성공적으로 배송한 package를 알기 위해선 먼저 이 트럭에 의해 이동되었거나 이동중인 package를 구해야한다.

tracking entity와 vehicle entity를 tracking\_id를 이용해 natural join 한다. 그후 truck의 정보를 이용해 사고난 트럭의 정보를 갖는 값들만 select한다. Select된 instance들의 package\_id 값(\*)은 해당 truck을 이용한 적인 있는 package들의 id이다. 그리고 tracking\_id(\*\*)는 이 트럭을 이용했을 때의 tracking 정보들을 갖는 id이다.

package\_id를 이용해 tracking entity를 grouping 한다. 그 후 아까 구한 package\_id(\*)를 이용하여 이 truck을 이용한 적이 있는 group만 남긴다. 그 후 각 group의 current\_time 값이 max 값을 갖는 instance 들의 조사해서 각 package들의 최신 tracking 정보를 얻는다. 이 최신 tracking 정보들 중 아까 구한 tracking\_id(\*\*)의 목록에 있는 package들은 사고가 날 당시 트럭에 속해 있는 package 들이기에 지워준다. 그후 남은 tracking 정보들 중에 current\_time이 가장 큰 값을 instance의 package\_id 값을 구하면 그 package\_id가 이 truck이 사고 나기전 마지막으로 성공한 delivery이다.

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

Payment entity를 customer\_id attribute를 이용해 grouping한다. 이때 year의 값은 전년도의 값을 갖는 항목들로 제한해준다. 그 후 count(package\_id)를 이용해주면 각 customer 당 보내거나 받은 package의 개수를 알 수 있다.

이를 (customer\_id, package\_nubmer) column을 갖는 새로운 table로 만들어준 후 package\_nubmer 값이 max 값을 갖는 customer 을 select해주면 된다. 앞서 year 값을 전년도 값을 갖는 항목들로 제한 해주었기에 작년에 package 개수가 최대인 고객을 알 수 있다.