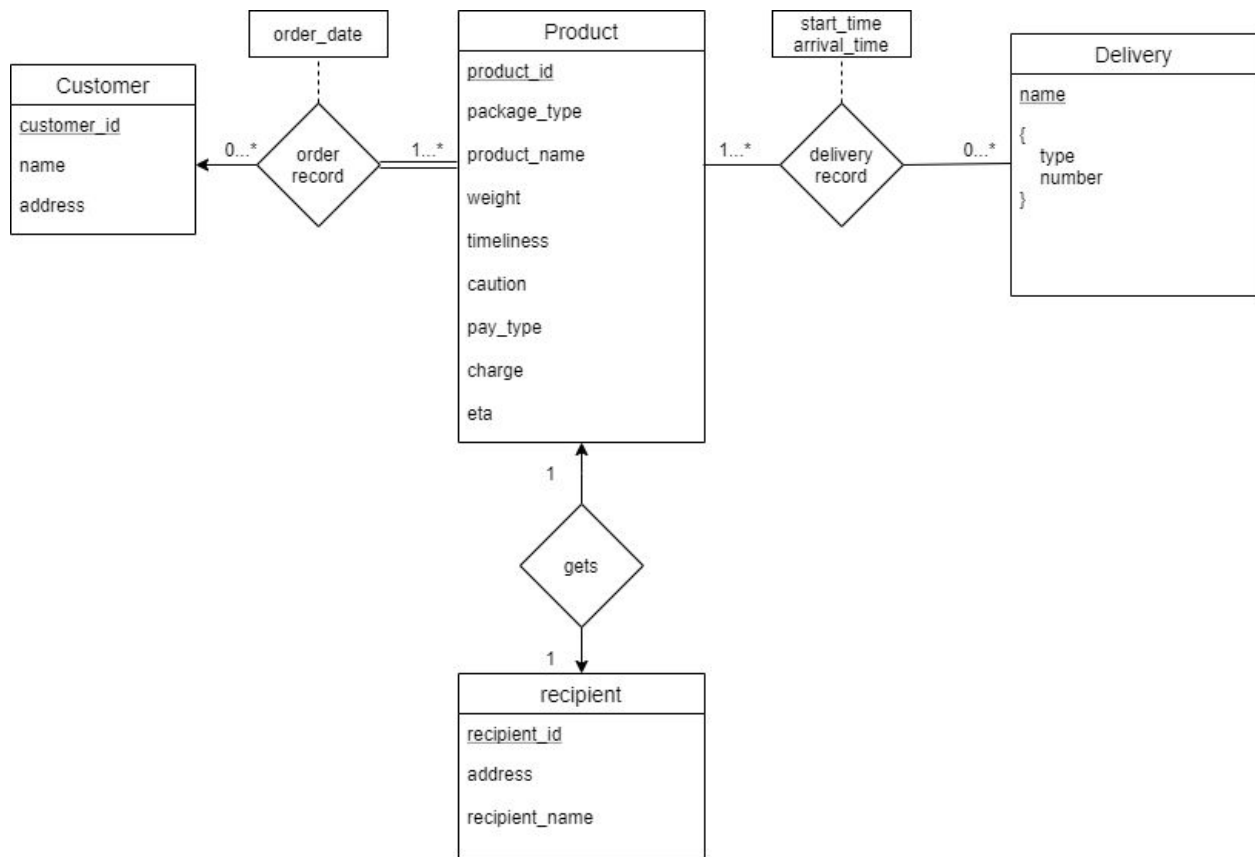


# Project 1 - Package Delivery System Report

20151550 박태준

## 1. E-R Model



### 1.1 Entity 설명

Customer : 고객에 대한 정보를 담고 있는 Entity.

customer\_id : 고객의 고유 번호

name : 고객의 이름

address : 고객의 주소

---


배송 업체에 Customer를 생각하면 보통 송신인과 수신인, 총 두 명을 생각하게 됩니다. 그러나, 송신인과 수신인 두 경우를 나눠서 데이터베이스를 고안할 경우, 아래와 같은 문제가 발생합니다.

- 일반적인 경우 송신인이 요금을 낸다.
- 착불의 경우 수신인이 요금을 낸다.
- 배송 업체가 송신인, 수신인에게 같은 정보를 제공하지만 따로 개별적으로 정보를 제공해야 한다.
- 수신인이 배송 업체에 가입되지 않을 수 있다.

따라서 이를 해결하고자 두 경우를 나누지 않고, 송신인을 Customer로 규정합니다.

이는 실제로 데이터를 처리하는 방식이라 생각해 적용하였는데,



 운송장 번호 추적

고객님의 운송장번호를 통해 요청하신 물품의 현재 위치 및 반품결과를 확인하실 수 있습니다.  
운송장번호는 13자리 숫자로 표기되며, 기입하실 때는 숫자 사이의 "-" 를 제외하고  
기입해주시기 바랍니다.

운송장번호

조 회

일례로, 구글 스토어에서 국내 배송을 할 때, 현대 운송을 이용하는데, 해당 운송업체로부터 배송 정보는 구글 스토어가 받고, 이를 구글 스토어가 다시 구글 스토어 구매자에게 제공하는 방식을 사용합니다. 이처럼 결제 제품에 대해 배송 조회를 할 경우, 배송비를 지불한 회사에서 배송 업체로부터 배송 product\_id 를 받고, 이를 수신자에게 연락해 확인하는 방법을 사용하고 있습니다. 즉, 배송 업체로서는 두 수신자를 고려하기보다, 결제인을 기준으로 정보 제공을 하는 것이 바람직하다고 판단했습니다.

Product : 고객의 배송요청에 대한 정보를 가진 Entity.

---

product\_id : 배송 업체의 DB이기에 각각의 배송 요청을 하나의 제품으로 보았습니다.

package\_type : flat envelope, small box, larger boxes 를 구분하기 위한 attribute.

product\_name : 요청 제품의 이름. 이는 product\_id 만으로는 어떤 배송이었는지 구분할 수 없는 고객을 위한 명시적인 요소로 사용됨.

weight : 배송 물품의 무게

timeliness : overnight, second day, longer 를 구분하기 위한 attribute.

caution : 제품에 특이 사항이 붙은 경우 표시하기 위함. ( hazardous, international ...)

pay\_type : 어떤 방식으로 값을 지불하는지 표시하기 위한 attribute.

charge : 해당 배송으로 청구되는 가격

eta : estimated time arrival 의 약어로, 예상 도착 시간을 의미합니다. 보통 물건 배송 요청시 결제 이전에 eta가 표시되기 때문에 각 product마다 예상 도착 시간을 각각 가진다고 생각했습니다.

Recipient : 배송을 받아야 하는 수신인을 나타내는 entity

recipient\_id: 수신인을 구분하기 위한 id

recipient\_name : 고객의 이름

address: 배송되어야 하는 주소

Delivery : 배송되면서 거치는 모든 요소를 나타내는 entity

예를 들어, 택배 집하장, 비행기, 트럭, 선적, 항구 등 배달물품이 머무를 수 있는 모든 곳을 포함합니다.

delivery\_id : 해당 요소의 고유 번호

name : 해당 요소의 설명이 들어가있습니다. (ex. truck 1721, center 51, plane 219)

type : 운송 요소의 종류 (ex. 트럭, 집하장, 비행기, 수취인(배송완료라는뜻))

---

number : 운송 요소 중복을 피하기 위한 번호

※ 실제 운송 회사의 데이터베이스의 경우 운송 요소를 더 세심하게 나눌 필요가 있습니다. 직접 운송수단을 관리, 운영하는 경우나, 운송 요소 별 처리를 요구하는 경우가 필요할 수 있기 때문입니다. 즉, name에 들어갈 요소를 또한 나눠 더 세심한 관리를 해줄 필요가 있으나, 이번 프로젝트는 name을 나누지 않아도 모든 요구사항을 만족시킬 수 있을 뿐만 아니라, Delivery\_record에 FK가 name 한 개가 추가되는 것과 type, number 두 개의 attribute가 추가되는 것이 수백 만 개의 row를 검사할 때 어느정도 차이가 발생할 수 있다고 생각하였습니다. 따라서 Relational Schema Diagram에서는 두 값을 합쳐 string으로 만든 name attribute만을 사용합니다.

## 1.2 Relationship 설명

order\_record:

Customer와 Product가 one-to-many 관계를 갖는 relationship입니다. 한 개의 Product는 여러 개의 Customer를 가질 수 없습니다. Product를 하나의 주문이라고 정의했기 때문입니다. 여러 명의 Customer가 같은 주문을 할 수는 없습니다. 그러나 Product의 경우, customer가 주문을 하지 않을 경우 존재할 수 없습니다. 따라서, Product는 모든 경우가 customer와 연관되어 있는 total의 관계를 갖음을 알 수 있습니다. 또한 회사에 가입만 하고 어떤 배송도 아직 하지 않은 고객이 존재할 수 있기에 order\_record에서 customer는 zero ,one or more 의 cardinality을 가지게 되고, 고객이 주문한 적 없는 배송은 존재할 수 없기에 product는 one or more의 cardinality를 가집니다.

order\_record relationship은 order\_date 라는 attribute를 갖는데, 이는 주문한 날짜를 의미합니다.

delivery\_record :

Product와 Delivery가 many-to-many 관계를 갖는 relationship 입니다. delivery는 배송되면서 거치는 모든 요소이기 때문에, 아래와 같은 예시로 설명할 수 있습니다.

배송하려는 제품 중 id 51에 대한 delivery\_record는 아래와 같습니다.

product_id ("배송51의 id") name ("truck 1571") start_time("2020-04-25 07:00") arrival_time("2020-04-25 16:00")	product_id ("배송51의 id") name ("port 3") start_time("2020-04-25 16:00") arrival_time("2020-04-25 20:00")	product_id ("배송51의 id") name ("ship 2") start_time("2020-04-25 20:00") arrival_time("2020-04-27 03:00")
--	--	--

start\_time 은 해당 운송 요소에 들어온 시각, arrival\_time은 해당 운송 요소를 떠난 시각을 의미합니다. 이처럼 해당 제품의 운송 기록들이 delivery\_record 로서 기록됨을 알 수 있습니다. 위의 예시를 통해 delivery가 many의 요소를 갖는 것을 알 수 있습니다.

product_id ("배송51의 id") name ("truck 1571") start_time("2020-04-25 07:00") arrival_time("2020-04-25 16:00")	product_id ("배송54의 id") name ("truck 1571") start_time("2020-04-25 07:00") arrival_time("2020-04-25 16:00")	product_id ("배송5922의 id") name ("truck 1571") start_time("2020-04-25 07:00") arrival_time("2020-04-27 16:00")
--	--	--

또한 위의 예시처럼 같은 트럭에서 서로 다른 제품들이 함께 배송되는 케이스가 존재할 수 있음을 알 수 있습니다. 위의 예시를 통해 product 또한 many의 요소를 갖는 것을 알 수 있습니다.

그렇다면 start\_time과 arrival\_time은 둘 중 하나만 표시해도 되는 것이 아닐까요? 어떤 요소의 arrival time은 다른 운송 요소의 start\_time이 될테니까요. 하지만, 두 요소는 명세사항에 의해 모두 존재해야 함을 알 수 있습니다. 명세사항 중

*Assume truck 1721 is destroyed in a crash.*

에 따라서, arrival\_time은 존재하지 않을 수 있습니다. 따라서 start\_time으로 분류하는 것이 바람직합니다. 그러나, start\_time만 확인한다면 배송이 완료되었는지 알 수 없습니다. 하지만 예를 들어, delivery\_id 가 0인 delivery name을 "customer" 라고 설정하고, arrival\_time에 대해 not null 임을 확인하면 배송이 완료되었음을 추가적인 attribute 없이 확인할 수 있습니다. 그렇기 때문에 start\_time과 arrival\_time 모두 필요한 요소임을 알 수 있습니다.

cardinality의 경우, 배송은 접수하는 순간 delivery record가 기록된다고 가정하였습니다. 배송 시작지에서의 기록이 필요하기 때문입니다. 반면, delivery의 경우, 예를 들어, 새로 만든 지점이 아직 갓 개업한 경우 아직 어떠한 배송 요청도 업무도 받지 못했다고 가정하여서 zero,one or more의 cardinality를 가진다고 가정하였습니다.

gets :

recipient gets product 라는 관계를 설명하기 위한 relationship 입니다. 한 배송 상품마다 한 도착지가 존재하기 때문에 one-to-one relationship을 갖습니다.

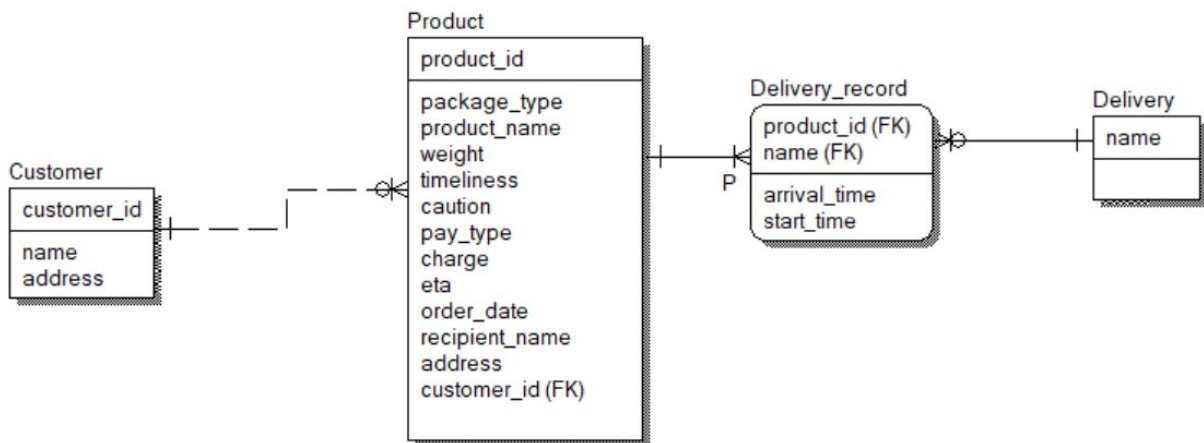
---

명세사항 중

*Find all recipients who had a package on that truck at the time of the crash.*

에 따라서, recipient에 대한 entity가 있는 것이 바람직하다는 생각 하에 위와 같이 정의하였습니다.

## 2. Relational Schema Diagram



위의 E-R 모델을 Relational Schema 로 만든 형태입니다.

### 2.1 Customer - Product

one-to-many 관계를 가지던 order\_record는 total 관계를 가진 many entity였던 Product의 attribute로 바뀌게 되었습니다. 따라서 order\_record 자체는 사라지고 customer의 primary key인 customer\_id를 attribute로 가져옵니다. 또한 order\_record의 attribute였던 order\_date 또한 product의 attribute가 됩니다.

고객이 배송 업체에 가입했으나 아직 아무런 주문을 한 적이 없다면 customer\_id에 해당하는 주문 내역이 존재하지 않을 수 있습니다. 따라서 두 테이블 사이에는 Zero,One or More cardinality가 허용됩니다. 또한 customer과의 관계인 order\_record는 primary key로서 역할을 하지 않기 때문에 Identifying relationship이라 할 수 없습니다. 따라서 두 관계는 non-identifying relationship의 관계를 갖습니다.

---

identifying relationship은 아니지만 모든 product는 고객의 주문으로부터 나오기 때문에 항상 customer\_id를 가지고 있어야 합니다. 따라서 null은 허용되지 않습니다.

## 2.2 Product - Delivery\_record

반면, many-to-many 관계를 가졌던 delivery\_record의 경우 table의 형태를 갖게 되었습니다.

delivery record 내에서 Product\_id는 foreign key이자 primary key로써 사용하기 때문에 두 관계는 identifying relationship이 되었습니다.

또한, 주문 즉시 배송을 시작하는 곳에 대한 table이 쓰여진다고 가정해 cardinality가 One or More로 설정하였습니다. 예를 들어, 배송 주문을 받는 A 지부에서 물품을 받아 배송주문이 들어간 순간, 아래와 같은 데이터가 삽입된다고 가정한 것입니다.

product_id("배송 물품의 id")
delivery_id("A 지부의 id")
start_time("접수 날짜 및 시각")
arrival_time(NULL)

## 2.3 Delivery\_record - Delivery

many-to-many 관계를 가졌던 delivery\_record의 경우 table의 형태를 갖게 되었습니다.

delivery record 내에서 name은 foreign key이자 primary key로써 사용하기 때문에 두 관계는 identifying relationship이 되었습니다.

cardinality에 대해서, 어떤 운송 요소가, 예를 들면 새로 건설한 집하장에 대한 정보가 DB에 추가되었을 때, 해당 운송장을 지나치지 않을 수 있다고 생각했습니다. 따라서, zero또한 포함되는 관계라 생각해 Zero,One or More로 설정하였습니다.

## 2.4 Product - Recipient

one-to-one 관계를 가졌던 두 entity가 합쳐질 때, 두 table이 1대1 관계를 가지기 때문에, 단순히 새로운 attribute를 추가하는 것만으로 조건을 만족시킬 수 있습니다. 따라서 product와 recipient관계는 합쳐졌습니다. recipient에 attribute들이었던 recipient\_id, recipient\_name, address는 모두 product의 속성에 들어오고, recipient를 구분하기 위해서 사용했던 recipient\_id는 더이상 사용하지 않게 되었습니다.

---

### 3.요구사항 명세서 검증

그렇다면 제시한 relational schema가 명세서에 적합한지 확인해보겠습니다. 즉, 명세서에서 요구하는 Queries를 직접 구현하면서 실현가능한지를 확인하는 것입니다.

쿼리에 대한 설명은 1번 쿼리에서만 진행하겠습니다.

#### 3.1 Queries

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.

2020년 4월 9일 truck 1721에서 사고가 났다고 가정시.

**select customer\_id**

**from product**

**where product\_id = (select product\_id from delivery\_record where name = "truck 1721" and start\_time = "2020-04-09");**

1721 트럭에서 사고가 난 경우, 먼저 당시 트럭에 있던 물건들을 찾고, 사고가 난 시간에 배송중이던 물건의 customer 를 찾아야 합니다. 따라서, 4월 9일에 사고가 났다면, truck 1721에서 4월 9일날 운송중이던 truck이 가지고 있던 product\_id를 delivery\_Record에서 가져옵니다. (select product\_id from delivery\_record where name = "truck 1721" and start\_time = "2020-04-09" )

이후, 이 product\_id 들과 매치되는 product 들의 customer\_id를 가져옵니다.

**select customer\_id from product where product\_id = ();**

2. Find all recipients who had a package on that truck at the time of the crash.

**select recipient\_name**



---

**from product**

**where product\_id = (select product\_id from delivery\_record where name = "truck 1721")**

**and start\_time = "2020-04-09");**

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

**select product\_id**

**from delivery\_record**

**where name = "truck 1721" and arrival\_time is not null**

**order by arrival\_time desc;**

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

**select max(count(customer\_id)),customer\_id**

**from product**

**where order\_date between "2020-04-30" and "2020-04-01";**

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

**select customer\_id,sum(charge)**

**from (customer natural join product) group by customer\_id**

**order by sum(charge) desc;**

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

**select p.product\_id**

**from product as p, deliver\_record as dr**

**where dr.product\_id = p.product\_id and dr.arrival\_time > p.eta;**

---

7. 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.

customer "1776019956" 에 대한 bill 만들기

simple bill :

```
select customer_name,address,sum(charge),month(order_date)  
  
from (customer natural join product)  
  
where customer_id = '1776019956'  
  
group by month(order_date);
```

bill listing charges by type of service :

```
select customer_name,pay_type,sum(charge),month(order_date)  
  
from (customer natural join product_id)  
  
where customer_id = '1776019956'  
  
group by pay_type,month(order_date)
```

itemize billing:

```
select product_name,charge  
  
from product  
  
where customer_id = '1776019956'
```