과목명: 데이터베이스시트템

<<Project #1>>

서강대학교 [컴퓨터공학과]

[20161566]

[권형준]

목 차

1. 프로젝트 구상

1.1 Entity 설명
1.2 Relationship 설명

2. ERD-Entity

2.1 Customer
2.2 Package
2.3 Bills
2.4 Transportation
2.5 Stops
2.6 Warehouse
2.7 Recipient

3. ERD-Relationship

3.1 Order
3.2 Receive
3.3 Store
3.4 Move

4. Logical Model

4.1 Customer-Package
4.2 Recipient-Package
4.3 Bills-Package
4.4 Package-Move
4.5 Package-Store
4.6 Store-Warehouse
4.7 Move-Transportation
4.8 Transportation-Stops

5. Query 적용

1. 프로젝트 구상

1.1 Entity 설정 기준

어떤 것을 entity로 설정할지의 기준은 크게 두가지였다. 첫째는 중복을 최소한으로 하게 entity를 정하는 것이었고, 둘째는 '그 Entity에 ID를 부여할 가치가 있는가?'였다. 후자의 의미는 Store, Move, Receive와 같은 행동, 현상에는 ID를 붙일 수 있지만 최대한 relation set으로 표현하려고 노력했다는 뜻이다. 각 Entity에 대한 자세한 설명은 뒤에서 하고 여기서는 각 Entity를 그것으로 정한 이유에 대해서 설명하겠다.

- Customer Entity

Customer entity는 가장 먼저 설정한 entity였다. 명세서의 나온 조건 속에서는, 'some customers have', 'Find all customers', 'Find the customer', 등과 같이 customer에 대한 정보를 요구하는 경우가 많았다. Relationship set 또는 다른 entity의 attribute으로 설정하기에는 customer가 포함해야 될 내용, 즉 attribute들이 너무 많았고, 중복이 발생할 것 같았다.

- Package Entity

Package Entity는 Customer Entity와 마찬가지로 가장 처음 생성한 entity이다. 프로젝트의 핵심 entity이며, 요구하는 모든 조건 속에서 등장을 하였다.

- Bills Entity

Bills entity는 Package entity와 one-to-one relationship이며 동시에 total-to-total relationship을 이루고 있다. 즉, 배송을 하나 주문하면 생기는 영수증 같은 개념으로 만들었다. 따라서 Bills의 attribute들을 Package의 entity로 가지고 와도 큰 문제는 없을 것이다. 하지만 일단 가독성이 좋으며, 이후에 다른 조건으로 인해 Bills와 Package를 분리해야 할 가능성이 있기 때문에 분리하였다. 예를 들어서 어떤 물건의 배송을 할부로 하는 경우, 한 Package당 하나 이상의 Bill이 존재해야 될 수 있다. (어디까지나 예시로 들었기 때문에 억지스러울 수 있습니다.)

- Recipient Entity

Recipient entity을 만든 이유는 Customer entity와 비슷하다. 다른 점은 Customer entity는 프로젝트 명세서에서 주어진 조건을 충족하기 위해 필수적으로 필요했던 반면, Recipient는 Package의 entity로 만들어도 프로젝트 결과에는 큰 영향을 미치지 못한다는 것이다. 그럼에도 Recipient entity를 만든 이유는, Recipient에 대한 다양한 정보들이 추가될 수 있고, 정보의 중복을 피하기 위해서이다. 다른 Package라도 같은 Recipient에게 갈 수 있고, Recipient에 대한 주소, 전화번호, 이름 등을 알아야 될 수 있다.

- Transportation Entity

Transportation Entity를 만든 이유는, 회사나 고객이 현재 자신의 Package의 위치를, 또는 과거에 자신의 Package가 있었던 위치를 알 수 있어야 된다는 조건 때문이다. 하나의 Package가 목적지에 도달하기 전까지 거칠 수 있는 Transportation은 여러 개일 수 있기 때문에 attribute이 아닌 entity로 설정하였다. Multivalued attribute으로 하지 않은 이유는, 그 Transportation의 출발 시간, 위치 등에 대한 정보 또한 저장해야 되기 때문이다.

- Warehouse Entity

Transportation Entity를 생성한 이유와 마찬가지로 Warehouse Entity 또한 Package의 현재 위치, 과거 위치를 추적하기 위해 사용하였다. Package가 배송과정에서 있을 수 있는 Warehouse는 Transportation과 마찬가지로 복수이기 때문에 entity로 설정하였다.

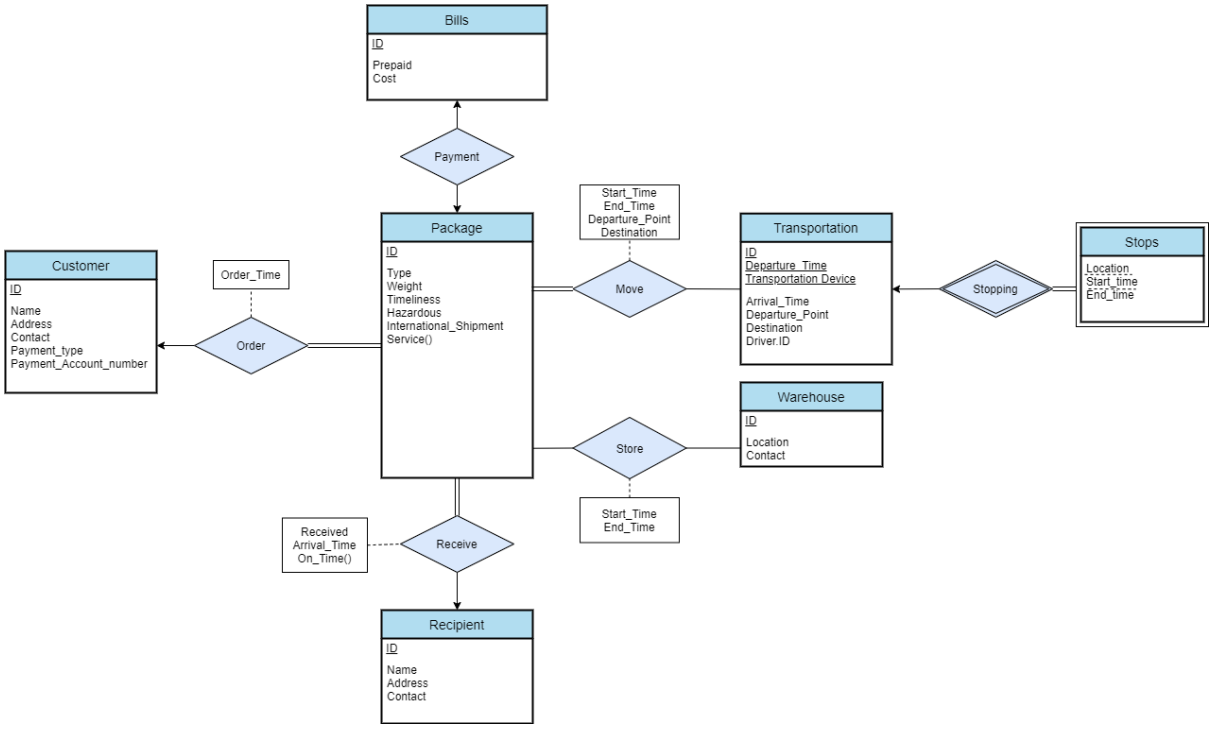
- Stops

Stops entity을 만들게 된 계기는 Transportation entity의 한계 때문이다. 운송차량이 배달을 할 때 가지고 있는 물품의 목적지가 다 같은 것은 아니다. 어떤 물건들은 중간에 목적지에 도달하거나 중간에 다른 운송 수단으로 전달이 된다. 그러는 도중 정차를 하게 될 것인데, 정차를 할 때마다 Transportation entity에 새로운 tuple을 추가하기에는 중복되는 정보의 양이 너무 많다. 따라서 transportation entity를 strong entity로 갖는 weak entity를 생성하고 정차하는 장소만 저장을 하는 Stops Entity를 만들게 되었다. 이렇게 되면 만약, '한번이라도 방배동을 방문한 차량을 구하여라.', 라는 조건이 있다면 출발지나 목적지가 방배동이 아니더라도 중간에 방배동에 정차한 기록이 있는 차량을 색출할 수 있다.

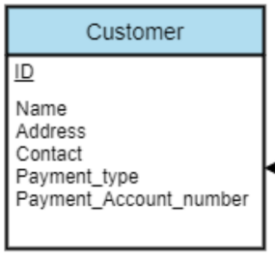
1.2 Relationship 설정 기준

Relationship은 관련 있는 Entity끼리 연결을 하고, 그때 발생하는 추가로 발생하는 정보들을 descriptive attribute로 추가하여 생성하였다. 각 relationship의 이름은 Customer 와 Package 사이의 Order 같이 서로를 연결 짓는 행동이나 현상으로 하였다. 각 relationship의 cardinality와 descriptive attribute은 뒤에서 살펴보도록 하겠다.

2. ERD-Entity



2.1 Customer



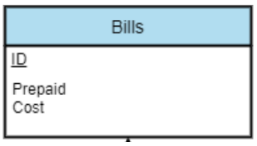
Customer Entity는 Customer에 대한 정보를 담고 있다. Key Attribute으로는 각각의

customer에게 고유하게 주어진 Customer.ID를 사용하였고, 이름, 주소, 연락처를 기본적으로 저장하였다. Address는 문제 조건에서 bill의 형태중 Simple bill을 표현하기 위해 필요하기 때문에 넣었고, Payment Type은 정기적으로 지불하는 customer와 그때 그때 지불하는 customer을 구분하라는 문제의 조건을 위해 추가하였다. Payment Account number는 정기적으로 지불하는 customer가 지불할 계좌번호를 저장하기 위해 추가하였다.

2.2 Package

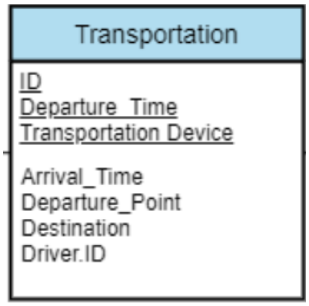
Package Entity또한 Customer와 마찬가지로 고유의 Package.ID를 이용하여 primary key를 설정하였다. 문제에서 주어진 조건 중, Package의 Type, Weight, Timeliness에 따라서 Service를 결정할 수 있다는 조건 때문에 각각을 attribute로 추가하였으며, Service는 Type, Weight, Timeliness로 인해 결정되기 때문에 Derived attribute로 표현하였다. 또한 Package의 내용은 신경 쓰지 않지만 위험한 물건이거나 해외 배송인 경우를 따로 처리한다고 하였기 때문에 해당하는 항목의 attribute을 추가하였다.

2.3 Bills



Bills entity는 각 Package의 가격에 대한 정보를 담고 있다. Primary key로는 Bill고유의 ID를 사용하였지만, Package와 one-to-one의 relationship을 이루고 있기 때문에 Package의 ID를 가져와서 사용해도 무관하다. 어떤 배송은 미리 결제가 되었을 수 있다는 문제조건을 충족하기 위해 prepaid라는 attribute을 넣었으며, 각 배송의 가격을 저장하기 위해 cost를 넣었다.

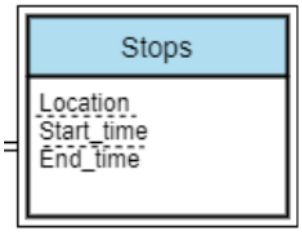
2.4 Transportation



Transportation Entity는 앞에서 설명한 것처럼 Package를 추적하기 위해 만들었다. Primary Key는 총 세가지로 구성 되어있다. 우선 비행기, 자동차, 기차, 배와 같이 운

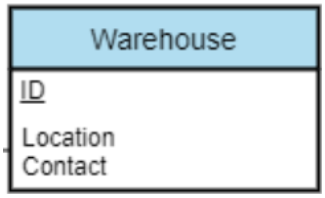
송수단을 저장한 Transportation Device, 만약 비행기라면 두 비행기를 구분하기 위해 각 운송수단마다 지정한 고유의 ID, 그리고 마지막으로 배송에 같은 운송수단이 중복되어 사용될 수 있기 때문에 Departure Time을 추가하였다. Departure time을 primary key로 넣지 못한다면 4시에 출발하여 부산에서 서울로 가는 KTX-934과 9시에 서울에서 부산으로 출발하는 KTX-934를 구분할 수 없고 추적할 없게 될 것이다. 하나의 운송수단이 출발시간이 같으면서 도착시간이 다를 수 없기 때문에 arrival time은 primary key에서 제외되었다. Departure point와 destination을 사용하여 특정 시간대에 특정 운송수단을 사용하고 있었던 Package들의 위치를 추적할 수 있고, Driver.ID는 이상이 있는 경우 연락을 해야 될 수 있기 때문에 추가하였다.

2.5 Stops



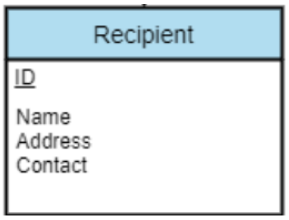
Stops Entity는 Transportation Entity의 불완전한 부분을 채워준다. 배송을 위한 운송수단은 한번 이동할 때 많은 Package들을 운반한다. 그 많은 Package들의 목적지는 같을 수도 있지만 다를 수 있다. 따라서 중간에 정차를 하는 구간이 생길 수 있는데, 이 때마다 Transportation tuple을 새로 생성하면, 정보의 중복이 크게 늘어나게 된다. 따라서 운송수단의 처음 출발지와 도착지만 Transportation entity의 attribute으로 설정하고 Stops라는 weak entity를 사용하여 중간에 정차하는 구간을 저장하게 된다.
Location, start time은 discriminator로 사용된다. Start time까지 포함된 이유는 가능성은 적지만 운송수단이 같은 곳에 두 번 이상 정차를 할 수 있기 때문이다.

2.6 Warehouse



Warehouse Entity는 Transportation과 마찬가지로 추적을 위해 생성하였다. 운송수단과 다르게 창고는 이동을 하지 않기 때문에 도착시간, 등으로 나눌 필요가 없었다. 단순하게 Warehouse 고유의 ID를 primary key로 설정하였으며, 위치와 연락처를 attribute로 추가하였다.

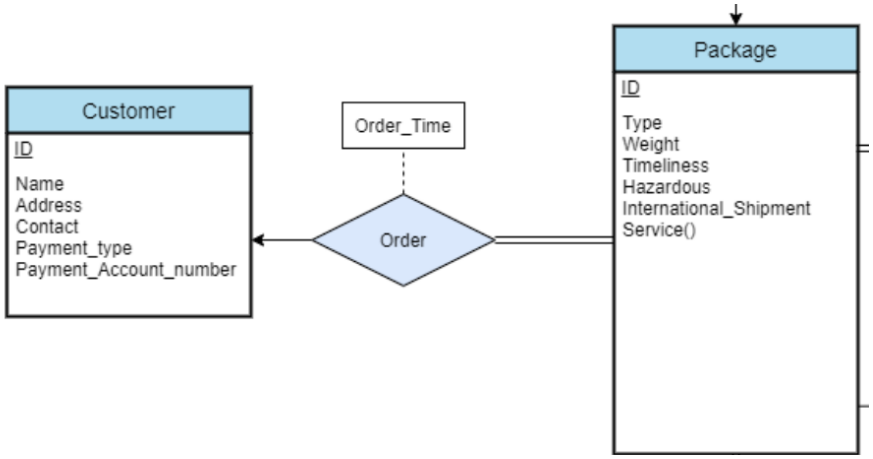
2.7 Recipient



Recipient Entity는 Package의 최종 목적지인 수취인의 정보를 저장한다. Package의 attribute으로 설정할 수 있었지만, 이름, 주소, 연락처 등 recipient에만 관련이 된 정보들이 많기 때문에 따로 entity를 생성하였으며, primary key로는 고유의 ID를 설정하였다.

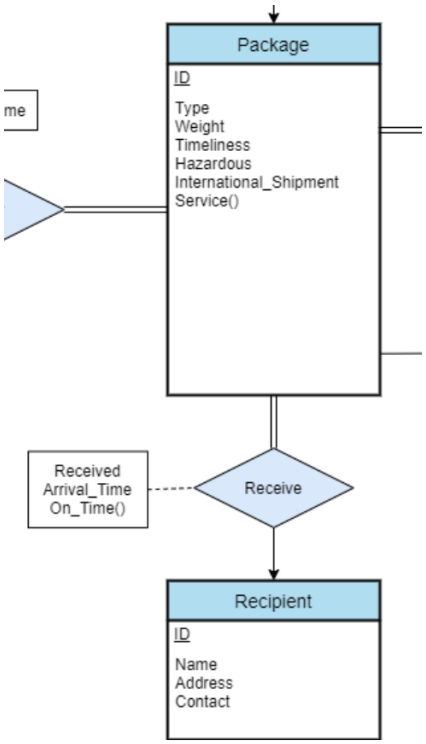
3. ERD-Relationship

3.1 Order



Order Relationship-set은 Customer와 Package를 연결하는 역할을 수행한다. 한 명의 Customer는 여러 개의 Package를 부칠 수 있고, 하나의 Package는 오직 한사람의 고객에 의해서 생성이 되기 때문에 one-to-many의 관계를 형성하고 있다. 또한 Package가 존재한다는 뜻은 주문한 사람, 즉 Customer가 존재한다는 뜻이기 때문에 Package는 이 관계에서 total이 된다. Descriptive attribute로는 Order time이 있는데, 이는 customer가 package를 주문한 시간(날짜, 시간, 분, 초)를 저장한 것이다. 이 attribute을 토대로 프로젝트 명세서에 나와있는 조건인 '재시간에 도착한 물품을 찾아라.', '지난 한달간…', '지난 1년간…' 등을 해결할 수 있다. 이 attribute은 나중에 reduction을 통해서 Package의 attribute으로 포함되게 된다.

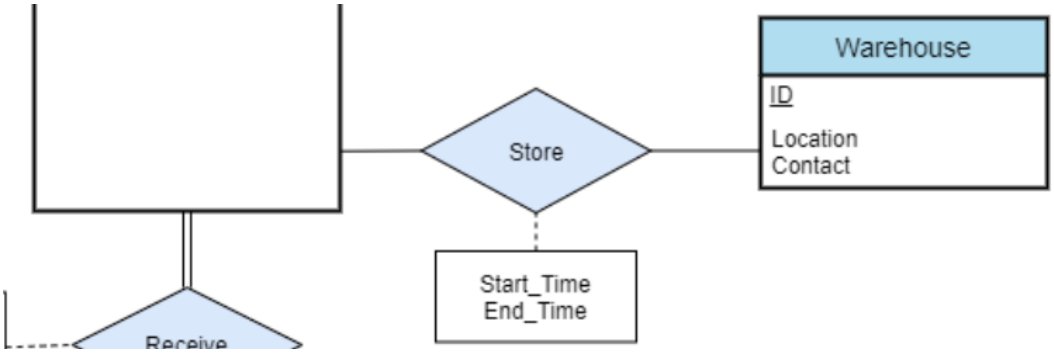
3.2 Receive



Receive는 Package와 Recipient를 연결한다. 하나의 Package는 최종 목적지가 한 곳이기 때문에 한 명의 Recipient에게만 연결이 되고, 한 명의 사람이 여러 개의 Package를 받을 수 있기 때문에 이 둘의 관계는 many-to-one이 된다. Customer와 마찬가지로 모든

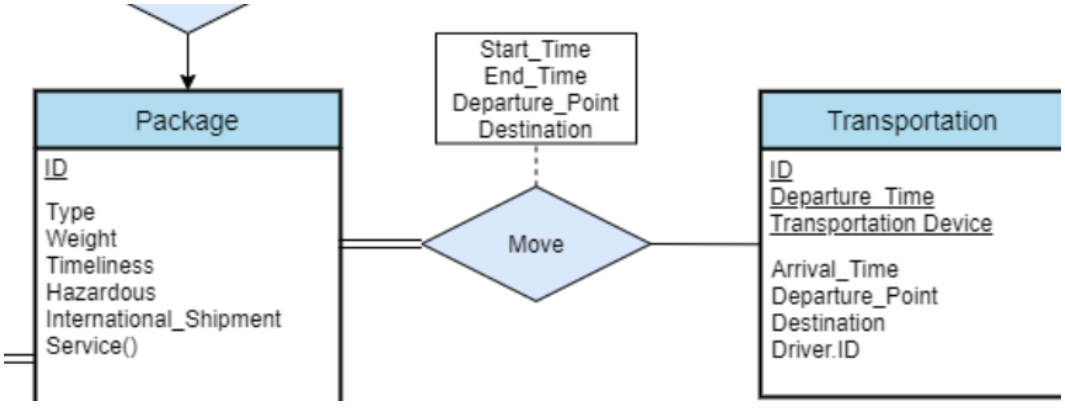
Package는 목적지(수취인)이 무조건 필요하기 때문에 package는 이 관계에서 total이 된다. Descriptive attribute으로는 정상적으로 Package가 도착했는지를 기록하는 Received, 도착 시간을 기록하는 Arrival Time, 재시간에 도착했는지를 기록하는 On Time이 있다. Received attribute은 Arrival time이 null인가, Package의 현재 위치가 Recipient의 Address와 동일한가 등으로 대체할 수 있지만 배송이 완료되었는지 확인을 해야 되는 경우가 많기 때문에 따로 attribute을 생성하였다. Received는 앞에서 말한 것처럼 다른 attribute으로 대체될 수 있는 만큼 derived attribute으로 설정할 수도 있지만 여기서는 그냥 도착하지 않았으면 no, 했으면 yes로 표현하는 독립적 attribute로 설정하였다. Arrival time은 도착시간을 기록함으로써 재시간에 도착했는지 확인하는 조건을 만족할 수 있고, 과거에 있던 Package의 경로를 추적하는데 도움이 된다. 마지막으로 On Time은 문제에서 제시한 조건인 '재시간에 도착하지 못한 Package를 고르시오'에 사용된다. On time은 order time, arrival time, Timeliness 세개의 attribute를 가지고 연산을 수행하여 derived 된다.

3.3 Store



Store의 좌측에 있는 Entity는 Package Entity이다. Store은 Warehouse와 Package Entity를 연결하며 하나의 창고에는 여러 개의 물품이 저장될 수 있고, 배송 과정에서 하나의 물품은 복수의 창고에 머무를 수 있음으로 many-to-many의 관계를 가진다. 또한 특정 물품은 배송 거리가 멀지 않아 바로 수취인에게 전달될 수 있음으로 꼭 창고에 머무르지 않아도 된다. 따라서 이 관계에서 total인 관계는 없다. 특정 물품의 추적을 위해 창고에서 머무르기 시작한 시간과, 창고에서 나오는 시간을 descriptive attribute으로 가진다.

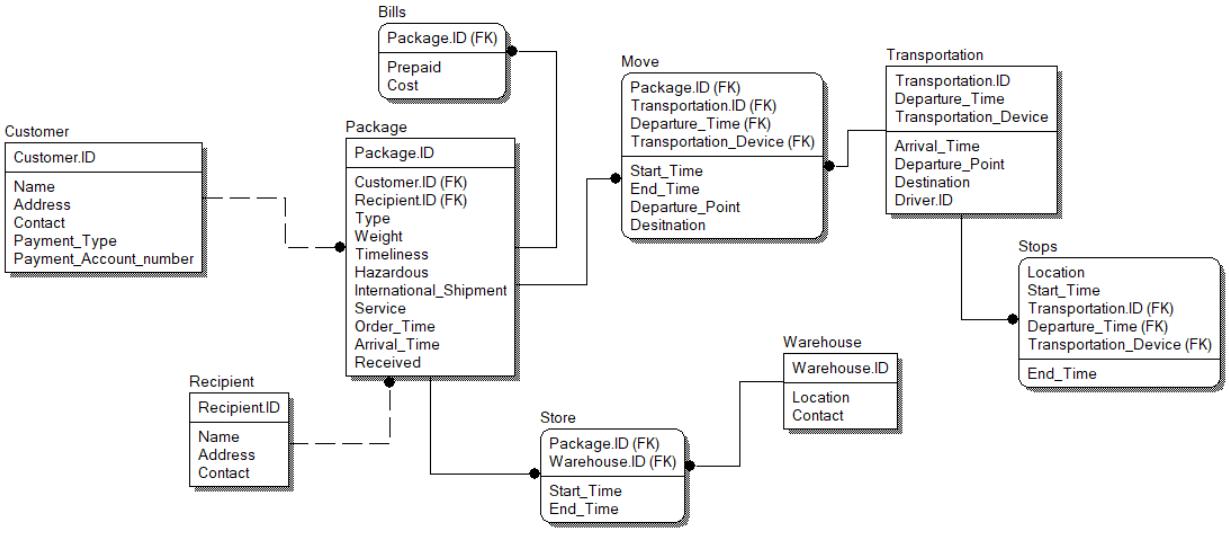
3.4 Move



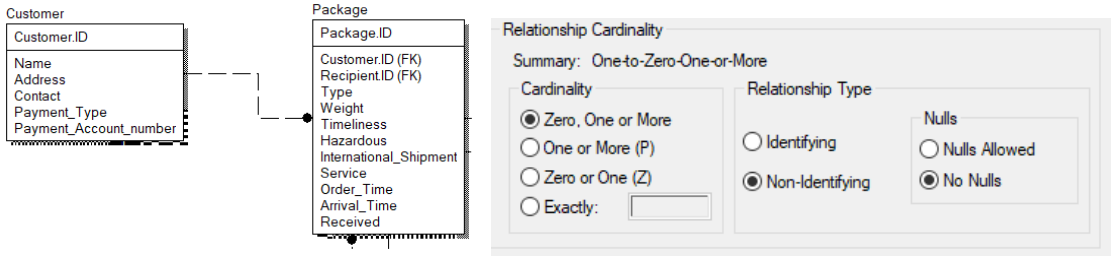
Move relationship-set은 Package와 Transportation을 이어준다. Package은 적어도 한번의 운송수단을 이용해야 되고, 하나 이상의 운송수단을 이용할 수 있다. 또한 운송수단은 한 번에 여러 개의 물품을 운송함으로 이 관계는 many-to-many이면서 package는 total이 된다. Descriptive attribute으로는 시작과 끝 시간, 시작과 끝 장소가 있다. Transportation에 있는 시작과 끝 시간, 시작과 끝 장소는 그 특정 운송수단의 시간과 장소를 뜻한다. 즉, 하나의 운송수단이 중간에 멈춰서 물품을 싣고 다시 이동할 수 있고, 중간에 창고다 목적지, 또는

다른 운송수단으로 물건을 전단 할 수 있다. 이런 점을 고려하면 Package별로 같은 Transportation을 이용하더라도 타는 곳, 내리는 곳, 타는 시간, 내리는 시간이 다를 수 있기 때문에 이러한 attribute들이 필요하다.

4. Logical Model

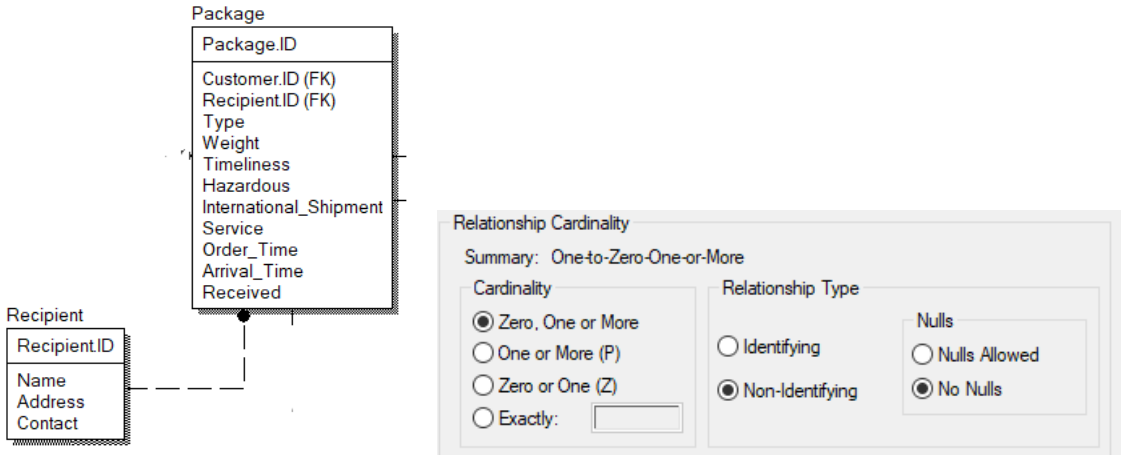


4.1 Customer-Package



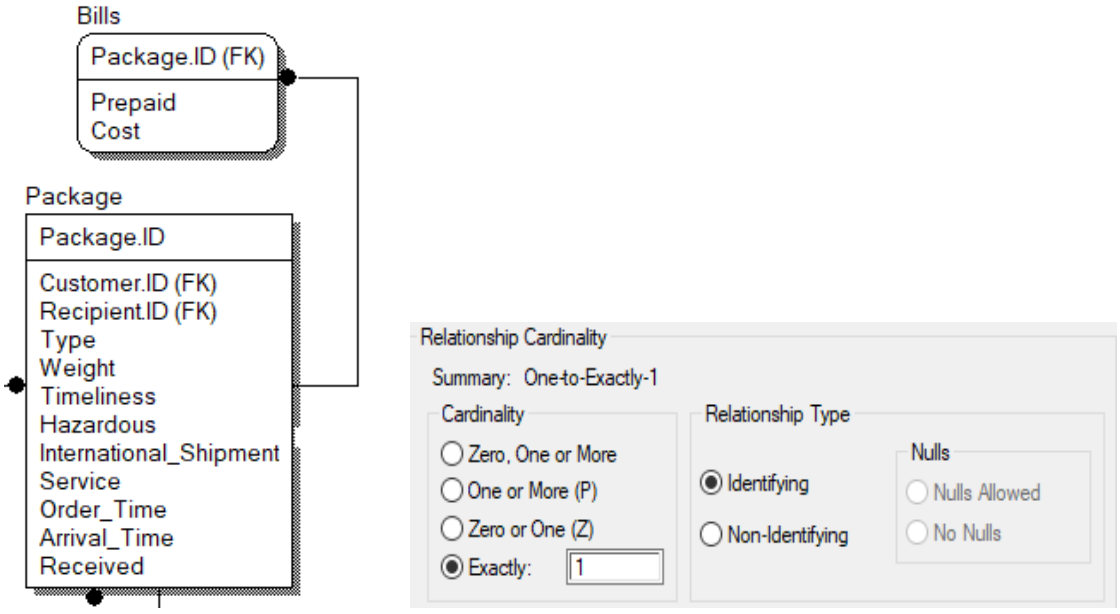
Customer과 Package사이에는 ER 모델 단계에서는 Order이라는 Relationship-set이 존재하였다. 하지만 Customer와 Package가 one-to-many이고 many 쪽이 total이었기 때문에 many쪽에 one쪽의 Primary key를 Froeign key로 넣는 것으로 relationship-set을 생략할 수 있다. 위의 Package entity를 살펴보면 Customer.ID가 Foreing key로 들어간 것을 확인할 수 있고, Order relationship-set의 descriptive attribute이 Package Entity에 포함된 것을 확인할 수 있다. Relationship properties를 확인해보면, ER 모델에서 Customer와 Package의 관계인 many(total)-to-one을 생각해보면 Customer은 Package를 0~n개까지 가질 수 있고, Package는 무조건 하나의 Customer을 가져야 되기 때문에 One-to-Zero,one or more이 됨을 알 수 있다.

4.2 Recipient-Package



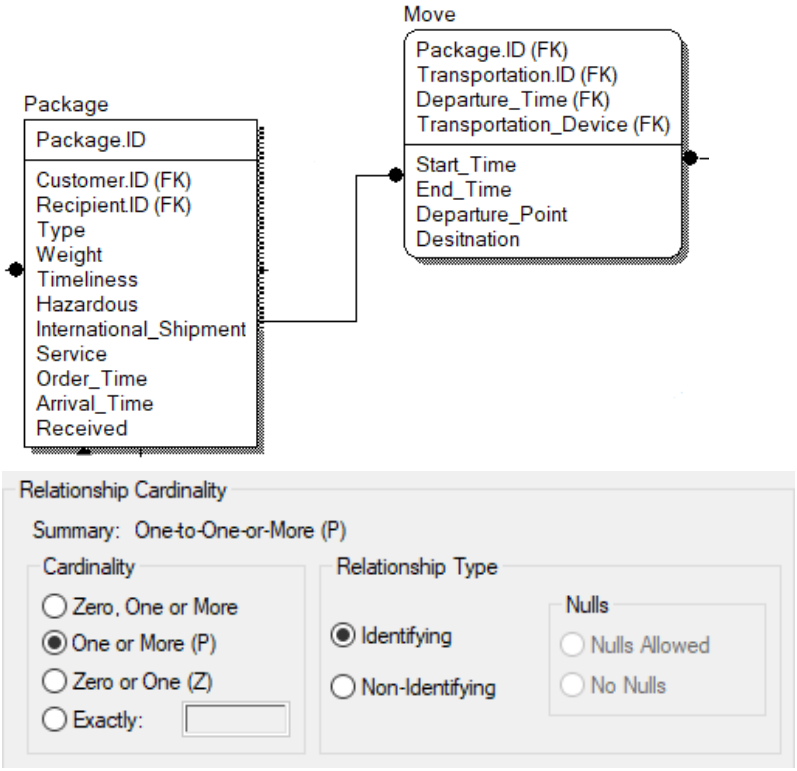
Recipient와 Package사이에는 ER 레벨에서는 Receive라는 relationship-set이 존재하였다. 하지만 이 relation은 many-to-one, 그리고 many 쪽이 total이었음으로 many쪽인 Package에 one 쪽의 Primary key인 Recipient ID를 추가하는 것으로 Relationship-set을 생략할 수 있다. 또한 descriptive attribute이었던, Arrival time, received는 Package에 추가되고 derived attribute인 on time은 생략된다. Relationship Cardinality를 살펴보면 Many(total)-to-one, 즉 one-to-zero, one or more임을 알 수 있다.

4.3 Bills-Package



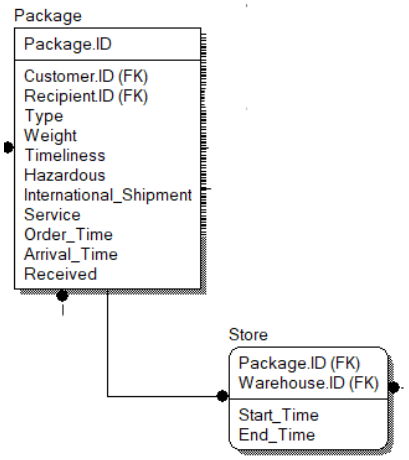
Bills와 Package의 관계는 Payment라는 relationship-set을 중심으로 one-to-one relationship이다. 또한 package가 있으면 bill이 한 개 존재해야 되며, bill 한 개당 package가 한 개 존재해야 된다. 이는 Bill.ID를 없애고 Package.ID를 Primary key이자 foreign key로 사용할 수 있게 함을 뜻한다. Relationship Cardinality를 확인하면 One-to-Exactly 1으로 1:1 대응임을 알 수 있다. 또한 Pcakge의 Primary key가 bill의 primary key로도 사용이 되었음으로 이 관계를 identifying relationship이라고도 할 수 있다.

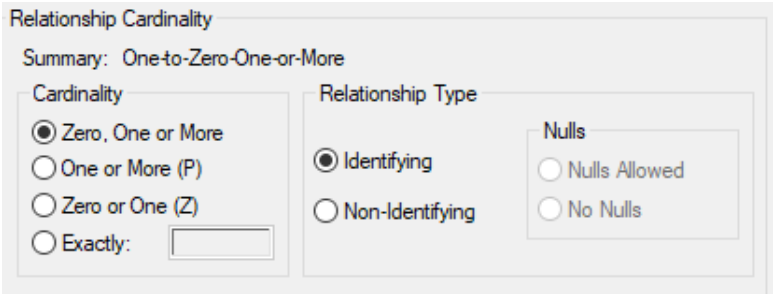
4.4 Package-Move



Move는 ER Level에서 Package와 Transportation entity를 연결해 주던 relationship-set에 속한다. Transportation과 package가 many(total)-to-one 또는 Strong과 weak entity가 아님으로 이 relationship-set은 생략할 수 없다. 따라서 양쪽 entity의 primary key를 가지고 와서 자신의 primary key로 삼고, descriptive attribute들을 attribute으로 갖는다. Package의 primary key인 Package.ID가 Move의 primary key로 사용되었음으로 이 관계 또한 identifying relationship에 속한다. Package 와 move는 ER모델에서 본 것과 같이 total이자 many의 관계이다. 애초에 Identifying relationship이기 때문에 null이 허용이 안됨으로 좌측이 zero가 허용이 안되고, total의 관계이기 때문에 우측 도한 zero가 허용이 안된다. 따라서 one-to-one or more의 관계를 가진다.

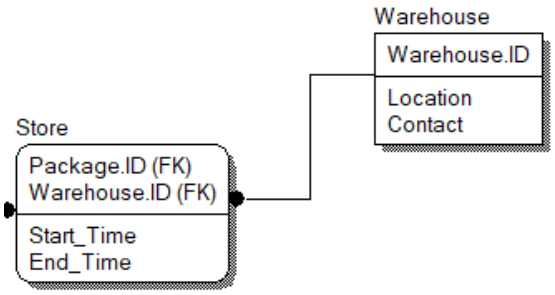
4.5 Package-Store

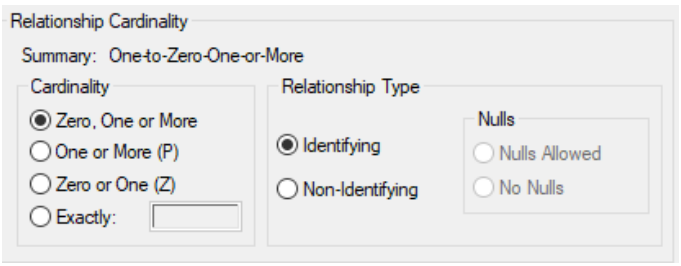




Store은 ER Level에서 Package와 Warehouse 사이의 relationship-set이었다. 이 relationship도 move처럼 생략될 수 없는 relationship임으로 양쪽 entity의 primary key를 primary key로 가지고 원래 가지고 있던 descriptive attribute을 기본 attribute으로 갖게 된다. 또한 이 관계는 total이 아니면서 package의 Primary key가 store의 primary key의 일부로 쓰이고 있기 때문에 identifying relationship이면서 zero가 허락된, one-to-zero, one or more의 관계를 가진다.

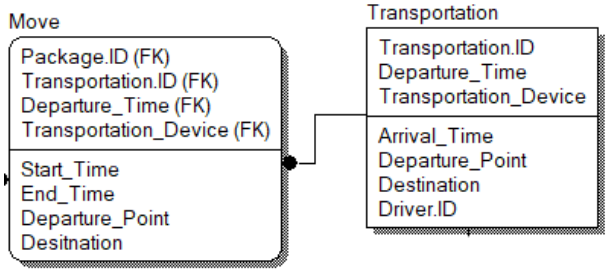
4.6 Store-Warehouse

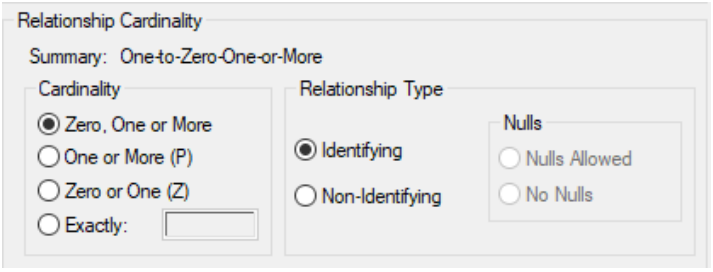




Store과 Warehouse relationship은 ER Level에서 many이면서 total은 아니었다. Package와 Store의 relationship과 동일함으로 One-to-Zero, one, or more 이 된다. Store Entity는 양쪽의 Entity에서 primary key를 가져와서 자신의 primary key로 설정함으로 Warehouse의 Primary key인 Warehouse.ID를 포함한다. 따라서 이 relationship도 identifying relationship이라고 할 수 있다.

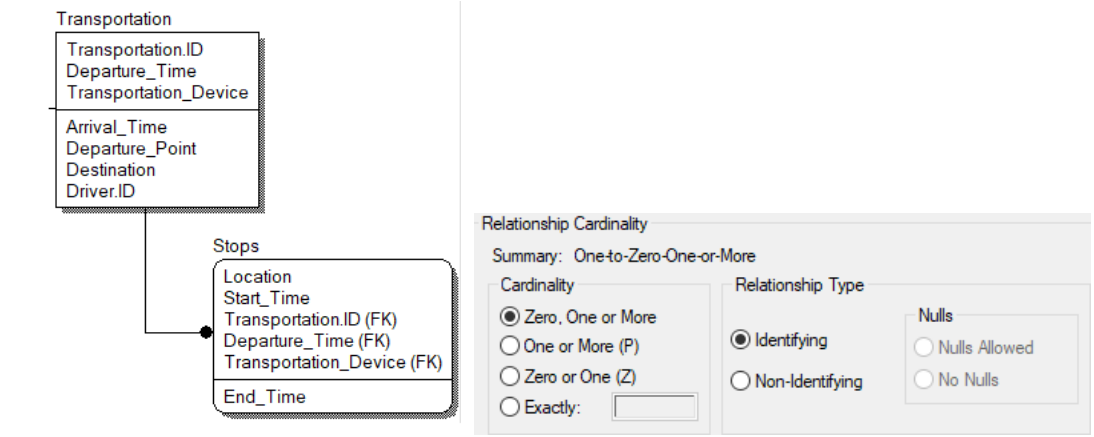
4.7 Move-Transportation





Transportation은 primary key가 총 3개이다. Move는 relationship set으로 양쪽 entity의 primary key를 가져와 자신의 primary key로 갖기 때문에 이 3개의 primary key를 모두 갖게 된다. 또한 그에 따라서 이 관계는 identifying relationship이다. ER Level에서 Package와 Transportation의 관계가 many(total)-to-many, 즉 한쪽은 total이 아니었기 때문에 Transportation 쪽에 zero가 포함된 one-to-zero, one or more의 관계를 갖는다.

4.8 Transportation-Stops



Stope Entity는 ER Level에서 Transporation의 weak entity였다. Weak entity는 reduction 과정에서 strong entity의 primary key와 자신의 discriminator를 primary key로 가진다. Transportation의 primary key를 primary key의 일부로 받았기 때문에 identifying relationship이며, Transportation은 Stop이 없을 수 있음으로 one-to zero, one or more의 관계를 형성한다.

5. Query 적용

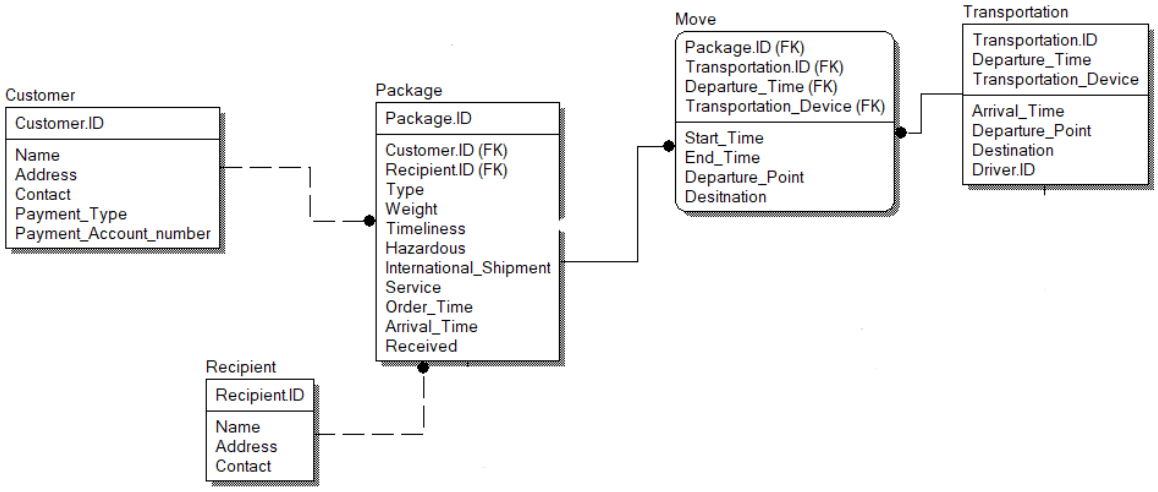
이제는 각 Query를 돌아보면서 어떤 방식으로 요구사항을 해결할 수 있는지 알아보자. 교재의 3단원에서 배운 SQL을 사용하여 식을 표현하였다. (SQL은 Logical schema를 바탕으로 만들었다. 또한 아직 안 배운 표현 방식이나, Domain이 미정이기 때문에 추상적으로 표현한 것들도 많다.)

5.1

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

다음의 Query를 해결하기 위해서는 Recipient, Move, Package, Customer, Transportation 총 4개의 Entity가 필요하다. 우선 Truck 1721과 사고 시간을 토대로 어떤 transportation을 살

펴볼 지 정한다. 다음 Move를 통해서 그 transportation을 사용한 Package들을 골라낸다. 골라낸 Package들이 사고 시점에도 그 차량에 있었는지는 Move의 start time과 end time의 사이에 사고가 일어난 시간이 있는 지로 판단할 수 있다. 그렇게 찾아낸 Package들은 Order 이나 Receive relationship으로 customer와 recipient들이 누구였는지 알아낼 수 있다. 그리고 사고 차량을 이용한 Package들 중 사고 시점 이전의 마지막 end time을 찾아서 last successful delivery를 알아낼 수 있다.



- Find all customer who had a package on the truck at the time of the crash(트럭의 출발 시간이 오후 1시이고 사고 시간이 오후 2시라고 가정)

```
select Customer.ID
from Package, (Select Package.ID as S
      from Move
      where Transportation.ID='1721'
      and Transportation_Device='Truck'
      and Departure_Time = '13:00'
      and start_time < '14:00'
      and end_time > '14:00')as T
where Package.Package.ID = T.S
```

우선 Move에서 조건을 충족하는 Package들을 색출한다. 다음 Package에서 앞의 조건을 만족하는 Package에 해당하는 Customer.ID를 색출한다.

- Find all recipients who had a package on that truck at the time of the crash(트럭의 출발 시간이 오후 1시이고 사고 시간이 오후 2시라고 가정)

```
select Recipient.ID
from Package, (Select Package.ID as S
      from Move
      where Transportation.ID='1721'
      and Transportation_Device='Truck'
      and Departure_Time = '13:00'
      and start_time < '14:00'
      and end_time > '14:00')as T
where Package.Package.ID = T.S
```

Customer와 마찬가지로 우선 사고 시점에 그 운송수단을 사용하고 있던 Package 들을 색출한 다음 그에 대응하는 Recipient를 색출한다.

- Find the last successful delivery by the truck prior to the crash(트럭의 출발 시간이 오후 1시이고 사고 시간이 오후 2시라고 가정)

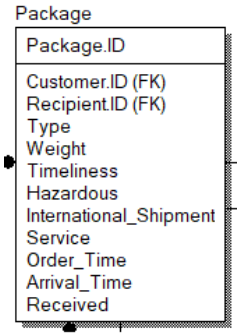
```
select Recipient.ID
from Package, (Select Package.ID as S, max(end_time)
      from Move
      where Transportation.ID='1721'
      and Transportation_Device='Truck'
      and Departure_Time = '13:00'
      and end_time <'14:00')as T
where Package.Package.ID = T.S
```

End time 이 사고 시간보다 이전이라는 것은 당연히 start time도 그 이전임으로 사고 이전에 그 운송수단을 사용한 Package라고 이해할 수 있다. 이때 max(end time)을 선택함으로써 마지막으로 성공적으로 배달한 Package를 찾을 수 있다.

5.2

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

이 query를 해결하기 위해서는 Package Entity를 살펴봐야 된다.



각 Customer가 주문한 Package는 모두 다 기록에 남으며, 지난 1년간이라는 조건을 충족하기 위해서 측정하는 시점의 현재 시간과 Package.Order_time을 비교하면 된다. 따라서 SQL로 작성하면 다음과 같다.

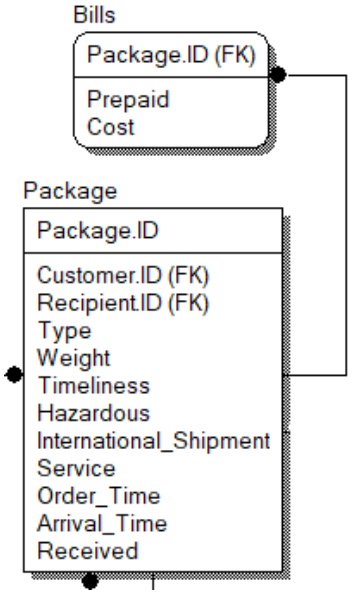
```
select Customer.ID
from (select Customer.ID, count(Customer.ID) as order_num
      from Package
      where (Order_time-Current_time)< '1 year'
      group by Customer.ID) as T
where T.order_num = (select max(order_num)
                     from T)
```

우선 Package에서 1년이내에 주문한 상품의 수를 Customer.ID를 기준으로 grouping하였다. 다음 그 group을 T라고 하고 그 T의 order num중에 최대를 만족하는 Customer.ID를 다시 찾았다.

5.3

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

이 Query는 직전의 query와 비슷하지만 Bill entity가 추가로 사용된다. 다음 schema를 살펴보자.



직전에 했던 query와 마찬가지로 1년 이내에 배송한 Package들을 Customer.ID별로 grouping을 하는데, 이번에는 count를 하지 않고 Cost의 합으로 두번째 attribute을 설정한다. 다음 그 값 중 최대값을 가지고 있는 Customer.ID를 색출한다. 이를 식으로 표현하면 다음과 같다.

```
select Customer.ID
from (select Customer.ID, sum(Cost) as total_Cost
      from Package, Bills
      where (Order_time-Current_time)< '1 year'
      and Bills.Package.ID=Package.Package.ID
      group by Customer.ID) as T
where T.total_Cost = (select max(total_Cost)
                      from T)
```

Bills와 Package를 Cartesian product한 뒤 양쪽의 Primary key인 Package.ID가 같은 것들만 고른다. 다음 Customer.ID로 grouping을 하는데 같은 Customer.ID의 Package에 해당하는 Cost들의 합을 total_Cost로 저장하였다. 다음 그 중에서 가장 높은 total_Cost를 지니고 있는 customer을 찾았다.

5.4

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

query를 만족하기 위해서는 Package Entity하나만 확인하면 된다. 재시간에 도착을 했는지에 대한 기준은 Timeliness이라는 attribute가 담고 있다. Timeliness에는 당일 배송, 다음 날 배송, 또는 그 이후 배송이 있으며, 이를 토대로 Order time과 Arrival time을 비교하면 재시간에 도착했는지 안 했는지 확인할 수 있다. 이를 토대로 SQL까지 작성하면 다음과 같다.

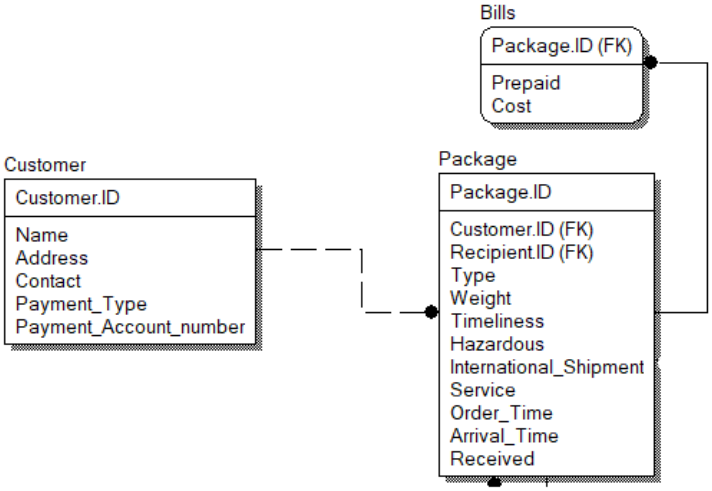

```
select Package.ID
from Package
where (Arrival_Time-Order_time) > Timeliness
```

이때 Timeliness와 Arrival, order time의 domain 등에 따라서 식이 달라질 수 있지만, 모두 다 연산이 가능한 숫자의 범위에서 Timeliness는 당일 배송이면 0, 다음날 배송이면 1 등으로 추상적으로 표현하였다. 나중에 Physical design의 단계에서는 이 식을 더 구체화할 필요가 있을 것이다.

5.5

- 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.

위의 3가지 종류의 bill을 다 표현하기 위해서는 총 3개의 entity가 필요로 한다. Customer, Package, 그리고 Bill entity가 사용이 된다.



- A simple bill: Customer, Address, amount owed

지난 한달 간의 기간동안 한 customer가 배송에 사용한 돈을 구하면 된다. 앞에서 1년 동안 가장 많은 돈을 사용한 고객의 SQL을 약간 변형하여 쓰면 다음과 같다.

```
select S.Customer.ID, S.Address, T.total_Cost
from Customer as S, (select Customer.ID, sum(Cost) as total_Cost
     from Package, Bills
     where(Order_Time-Current_Time)<'1 months'
     and Bills.Package.ID = Package.Package.ID
     group by Customer.ID)as T
where S.Customer.ID= T.Customer.ID
```

우선 Bills와 Package Entity를 가지고 최근 한달 이내에 각 Customer가 사용한 돈을 grouping 하고 그 다음 그 relation을 Customer와 다시 join하여 Address에 대한 정보까지 대응하는 customer에 맞게 출력한다.

- A bill listing charges by type of service

Package안에서 최근 한달 이내에 사용한 돈을 Customer와 service를 기준으로 grouping을 한 뒤 출력한다. 그 SQL은 다음과 같다.

```
select Customer.ID, Service, count(Cost) as total_cost
from Package, Bills
where (Order_time-Current_time)< '1 months'
and Bills.Package.ID=Package.Package.ID
group by Customer.ID, Service
```

이렇게 하면 각 customer에 대해서 service 별로 사용한 돈이 따로 출력된다.

- An itemize billing listing each individual shipment and the charges for it

가장 단순한 형태의 bill로 각 Package에 대한 가격을 표시하면 된다.

```
select Customer.ID, Package.ID, Cost
from Package, Bills
where (Order_time-Current_time)< '1 months'
and Bills.Package.ID=Package.Package.ID
```

감사합니다!