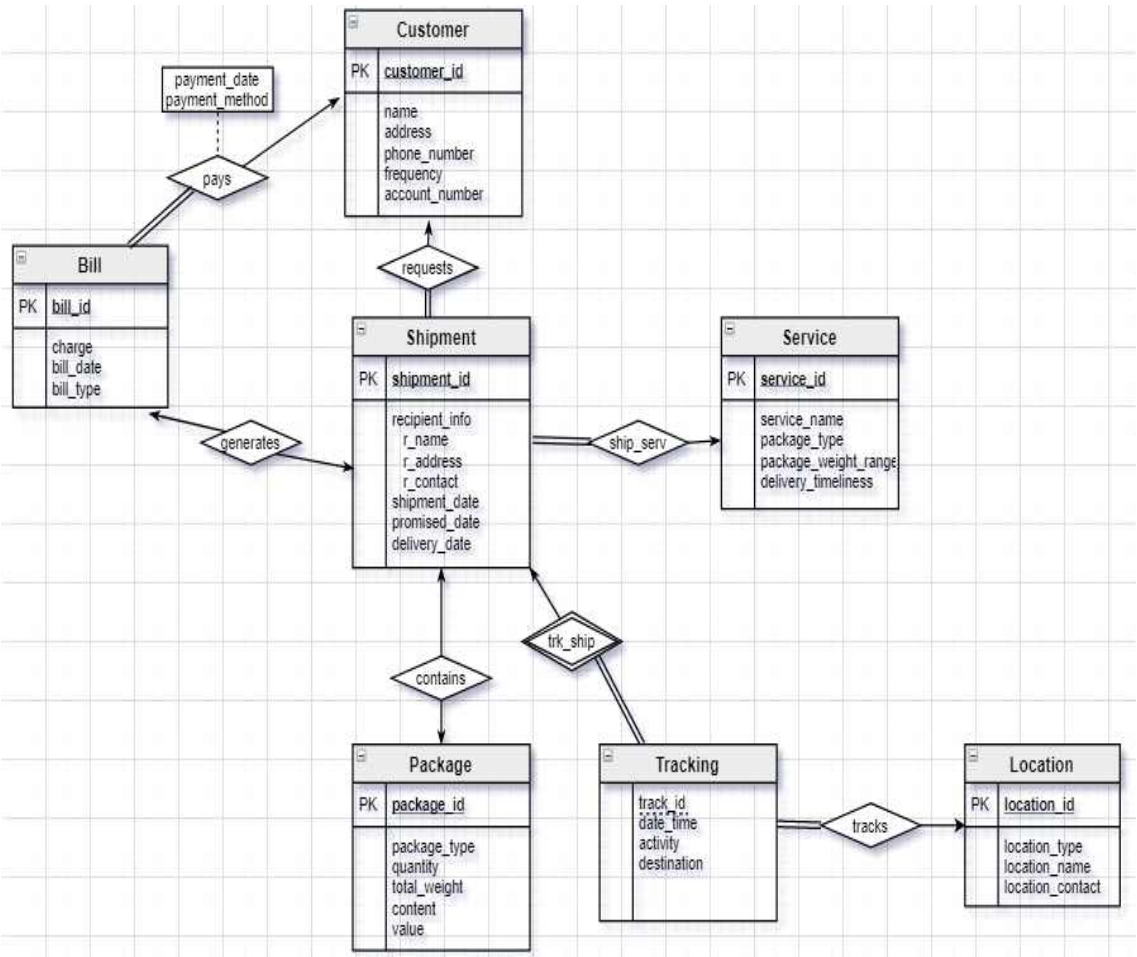
# Package Delivery System DB design 제안서

SG Company DBA
이 주형

안녕하십니까? 이번 DB 설계를 맡은 SG Company의 DBA 이주형이라고 합니다.
저는 초등학교 시절부터 농구 카페에서 활동하며 농구화 및 농구용품을 거래해왔습니다. 물품을 각각 100회 이상 부치고 수령해보았고, 송장번호 추적도 수없이 해보았습니다. 위와 같은 경험을 바탕으로, 저는 자사의 DBA로서 다음과 같이 package delivery system DB를 디자인했습니다.

[ER Diagram]



**I) 프로젝트 명세서를 통한 Entity, *Relationship* 설정 로직**

Application Description에 따르면, 이번 프로젝트에서의 DB디자인은 package handling, 그리고 Billing 에 대한 측면만 고려한다고 하였습니다. 따라서, 저는 우선 Package, Bill, 그리고 이러한 package를 보내고 Bill을 청구받는 고객 Customer 이렇게 세 가지를 우선 Entity로 설정하고 시작했습니다. 그리고 Customer-Package, Customer-Bill의 Relationship을 어떻게 할 것인지를 고민했습니다.

여기서 Customer가 Package를 보내는 Shipment는, 이번 DB 디자인의 가장 중요한 부분이라고 생각했습니다. Package handling과 Billing 모두 이 Shipment를 기준으로 용이하게 관리될 수 있기 때문입니다. Shipment를 Relationship으로 설정하기엔 너무나도 많은 정보가 반영된다고 판단했기에 Entity로 설정했습니다. 그리고 Shipment의 기간이나 가격 등을 결정하게 될 서비스의 종류를 Service 라는 Entity를 통해 별도로 관리하고자 했습니다.

Bill의 경우, Shipment와는 *generates*라는 Relationship을 통해 연결해주었습니다. 이를 통해 Bill에는 shipment 정보를 반영할 수 있게 됩니다. 이 때, Shipment가 이미 Customer의 ID를 FK로 갖기 때문에 사실 Bill과 Customer 사이엔 Relationship을 설정하지 않아도 처리가 가능합니다. *pays*라는 Relationship 설정을 통해 청구받은 Bill에 대한 고객의 결제가 최종적으로 완료되었는지 확인할 수 있도록 했습니다.

한편 프로젝트 명세서는 어떤 위치에 현재 물품이 위치하고 있는지, 이전에는 어디에 있었는지 등의 Location을 강조하고 있기에, 이를 Entity로 설정해주었습니다. 그리고 이러한 Location은 Shipment를 기준으로 Track하게 됩니다. 허나 Tracking에는 시간정보 뿐만 아니라 현재 및 과거의 위치 등 다양한 정보를 포함해야 하므로 Relationship 대신 Entity로 설정해주었습니다.

위와 같은 논리에 따라, 저는 주요 요소들을 *Relationship*보단 추가적인 Entity를 만들어 해결하고자 했습니다. 그리고 이러한 Entity들을 'Customer *requests* Shipment' 등 적절한 이름의 *Relationship*을 통해 각각 연결해주었습니다.

각 Entity와 Relationship, 그리고 그 속성 등에 대한 자세한 설명은 다음 파트에 이어서 진행하겠습니다.

## II) Attribute와 PK의 설정, 그리고 Cardinality 설정

다음은 각각의 모든 Entity와 일부 Relationship에 대한 attribute 설정, PK 설정, 그리고 Cardinality에 대한 설명입니다. 설명은 Entity에 대한 **attribute**와 **PK**설정, 그리고 Relationship에 대한 attribute와 Cardinality 설정 순으로 하겠습니다.

### Customer

고객의 정보를 관리해야 하기에, Customer entity를 만들었습니다. 여기서 고객은 자사의 delivery service에 돈을 지불한 'sender'에 해당하며, 간접 고객인 'receiver'는 포함하지 않습니다. 물론 sender에 해당하는 고객이 receiver일 때도 있겠지만, DB에 등록되지 않은 고객 혹은 주소로도 발송이 가능하도록 하기 위해서입니다. 고객의 기본 정보인 **name**, **address**, **phone_number**, 얼마나 자주 사용하는지를 나타내는 **frequency**, 그리고 **account_number**를 attribute로 추가해주었습니다. 이 때 동명이인의 고객이 존재할 수 있기에, **ID**를 고객마다 부여해 PK로 설정했습니다.

### Bill

청구 정보를 저장하기 위해 Bill entity를 만들었습니다. Bill은 각 Shipment에 대한 것으로 대응됩니다. 이 때, Bill은 청구되는 금액인 **charge**, 청구 날짜인 **bill_date**, 그리고 청구 방식인 **bill_type**로 구성했습니다. 여기서 bill_type은 고객의 정보를 반영해 월말에 함께 청구될 것인지, 이번 단일 건에 대한 credit_card 결제가 청구될 것인지, 혹은 prepaid된 것인지에 해당합니다. 이 때, prepaid의 경우 불량품이나 맞지 않는 옷 등을 고객이 반품할 수 있도록 회사에 이미 배송비가 선결제 된 것으로, 해당하는 Shipment에 대해선 prepaid인 경우 고객에겐 0원이 청구됩니다. PK로는 간단히 **ID**를 설정했습니다.

### Shipment

이번 디자인에서 가장 많은 relationship을 가짐으로써 가장 중요한 Entity입니다. package delivery 회사에서 배송정보를 관리해야 하기에 Shipment Entity를 생성했습니다. Shipment Entity에는 delivery에 대한 대부분의 사항이 반영된다고 보시면 되겠습니다.

Shipment에는 수령인의 정보가 속성으로 반영됩니다. 이 때, 수령인인 **recipient**의 속성은 **name(이름)**, **address(주소)**, **contact(연락처)**가 반영되게 됩니다. 또한 **shipment date**, **promised date**, **delivery date**를 통해서 접수일, 도착예정일, 도착일을 반영하도록 했습니다. 이를 통해 Shipment 중에 도착예정일보다 실제도착일이 늦은 경우의 query를 수행할 수 있습니다.

Shipment 역시 각각의 배송 정보를 쉽게 구분하기 위해 **ID**를 PK로 설정했습니다. 이 shipment_id는 추후 tracking 과정에서 송장번호(tracking number)의 역할을 하게 될 것입니다.

## Package

Package는 customer가 의뢰하는 일련의 물품 전체를 하나로 간주한 것을 의미하며, shipment 과정에 포함되는 대상이므로 Package entity를 만들었습니다. 관리를 보다 용이하게 하기 위해, 하나의 shipment에 여러 package를 보내는 경우, 각각의 package를 관리하는 대신 한 Shipment에 부치는 모든 package의 묶음을 하나의 큰 Package로 간주하고 여기에 **ID**를 부여하도록 했습니다. 즉 하나의 Shipment에는 ID를 부여받은 하나의 큰 Package가 대응되는 셈입니다.

그리고 여러 종류의 물품이 하나의 shipment에 포함 된 경우, 가장 부피가 큰 물품의 type을 기준으로 **package_type**을 설정합니다(e.g. flat envelope + small box -> small box). 실제로 타 업체에서도 큰 택배를 보내는 경우 가장 부피가 크거나 값비싼 상품을 기준으로 물품 종류와 가액을 기재하도록 되어 있기 때문입니다. 대신 몇 개의 작은 package가 포함되었는지 **quantity** 항목을 주어, 가장 많은 물품을 보낸 Customer를 계산할 때 활용할 수 있도록 하였습니다.

그리고 shipment에 포함된 모든 크고 작은 package의 무게 총합을, **total_weight**로 계산합니다. 마지막으로 package가 유실될 경우나 위험물질/국제배송에 해당할 경우 등에 대비하여 내용물과 가액을 나타내는 **content**, **value**를 추가해주었습니다. 구분이 편하기에 **ID**로 PK로 설정했습니다.

## Service

'A bill listing charges by type of service'를 생성하기 위해선 서로 다른 종류의 서비스를 구분하고 그에 따른 charge(비용)가 달라질 수 있도록 Service entity를 만들어야겠다고 생각했습니다.

다양한 service 종류에 package의 type과 weight, 그리고 delivery timeliness가 영향을 미친다는 고려사항에 따라 **package_type**, **total_weight_range**, 그리고 **delivery_timeliness** 를 attribute로 우선 포함해주었습니다.

타 경쟁사에서도 서비스 종류에 따라 FedEx First Overnight, UPS 2$^{nd}$ Day Air 등 다양한 이름을 사용한다는 것을 고려해, 저 역시 자사의 각 서비스마다 **service_name**과 ID를 부여했고, 구분이 편한 **ID**를 PK로 설정했습니다.

어떤 package가 속해있는지 파악한 후, 그 package의 종류와 무게를 고려한 후 고객의 delivery_timeliness까지 함께 반영한 것이 어떤 서비스인지 service entity에서 검색될 것입니다.

## Tracking

현재뿐만 아니라 이전에 어느 장소에 있었는지, 어느 루트를 거쳐 왔는지 등을 Tracking 하기 위해 Tracking entity를 생성했습니다. Tracking은 Weak Entity Set으로, 기본적으로 Shipment가 Tracking을 Own하게 되는 형태입니다.

**track_id**에는 1,2,3,4 등의 숫자가 들어가며 discrimator가 됩니다. 따라서 Tracking에서 track_id를 통해서 특정 Shipment 개체의 그 동안의 모든 track을 파악할 수 있고, 해당 shipment의 max(track_id)를 찾아서 그 개체의 최근 tracking 정보를 파악할 수 있습니다. Shipment의 PK인 Shipment ID 그리고 track_id가 결합된 형태가 추후 schema diagram에서 Tracking의 PK를 구성하게 될 것입니다.

또한 날짜와 시각이 포함된 **date_time**을 기록하게 하였습니다. datetime 도메인을 활용하면 되기에 year 등으로 세분화하진 않았습니다. 그리고 **activity**(처리현황)에는 FK로 받아오는 location_id에 따른 picked up 정보, Warehouse, Truck, Plane에 대한 IN/OUT 정보, 그리고 signed 정보 등 detail이 반영됩니다. destination에는 현재 향하고 있는 위치(where it is currently headed)가 나타나게 됩니다.

각 Tracking은 하나의 location_id를 가지게 되므로, 특정 Shipment의 모든 이동경로를 추적하기 위해선 해당하는 Tracking의 컬럼을 각각 참조하게 됩니다. 모든 경로를 누적해서 적지 않음으로서 duplicate를 줄일 수 있기에 이런 방식을 채택했습니다. 또한 대부분의 package delivery company에서 역시 한 column의 track마다 하나의 location이 반영되도록 한 점에서 영감을 얻었습니다. (e.g. '곤지암 Hub, 간선하차' , '곤지암 Hub, 간선상차')

## Location

위의 Tracking에 있어서 각 위치의 정보를 관리하고자 Location entity를 설정했습니다. **location_type**에는 warehouse, plane, truck 등이 해당되며, **location_name**은 특정 이름이나 번호 등이 설정됩니다. **location_contact**에는 연락처를 반영해 필요시 연락할 수 있도록 하였습니다. 그리고 location_type과 location_name을 조합한 location_id를 PK로 설정했습니다. 이 location_id를 통해 data를 location standpoint에서 검색하거나, 어떤 물품이 해당 location에 위치 중인지 파악할 수 있을 것입니다.

## *pays*

Customer와 Bill의 관계인 pays에는 **payment_date**를 통해 실제 결제일을 반영하도록 했고, **payment_method**를 attribute로 포함해 신용카드나 계좌이체, 현금결제 등을 표시하게 했습니다. 보다 자세한 결제정보를 관리하려면 따로 Payment entity를 만들어야 했지만, 이번 디자인은 Billing에 초점이 있기 때문에 단순히 해당 bill이 결제처리가 되었느냐의 여부만 체크했습니다.

Customer는 여러 번 Bill을 청구받고 지불할 수 있고, 특정 Bill은 Shipment 정보에 해당하는 한 명의 customer에게만 주어지게 됩니다. 따라서 Bill->Customer relationship은 다대일 관계로 설정했습니다.

*generates*

Shipment와 Bill의 관계에 해당합니다. 하나의 Shipment에 대해 하나의 Bill이 청구되도록 디자인했으므로, Shipment->Bill은 일대일 관계로 설정했습니다.

*requests*

Customer와 Shipment의 관계에 해당합니다. 하나의 Customer는 여러 Shipment를 request 할 수 있지만, 각 Shipment는 오직 한 명의 Customer만 갖게 됩니다. 따라서 Shipment->Customer relationship은 다대일 관계로 설정했습니다.

*ship_serv*

Shipment와 Service의 관계에 해당합니다. 각 Shipment는 하나의 Service 타입을 갖게 되지만, Service는 여러 Shipment에 나타날 수 있습니다. 따라서 Shipment->Service relationship은 다대일 관계로 설정했습니다.

*contains*

Shipment와 Package의 관계에 해당합니다. Package는 하나의 Shipment에서만 나타나게 되며, 디자인에 따라 하나의 Shipment에는 ID를 부여받은 하나의 큰 Package 개체만 대응됩니다. 따라서 Shipment->Package 관계는 일대일 관계로 설정했습니다.

*trk_ship*

Tracking과 Shipment의 관계에 해당합니다. 각 Tracking은 하나의 특정 Shipment에 대한 tracking 정보를 나타냅니다. 반면 Shipment는 여러 개의 Tracking을 가질 수 있습니다. 그리고 Tracking은 Weak entity이므로, 다대일 관계를 가져야 합니다. 이에 따라 Tracking -> Shipment는 다대일 관계로 설정했습니다.

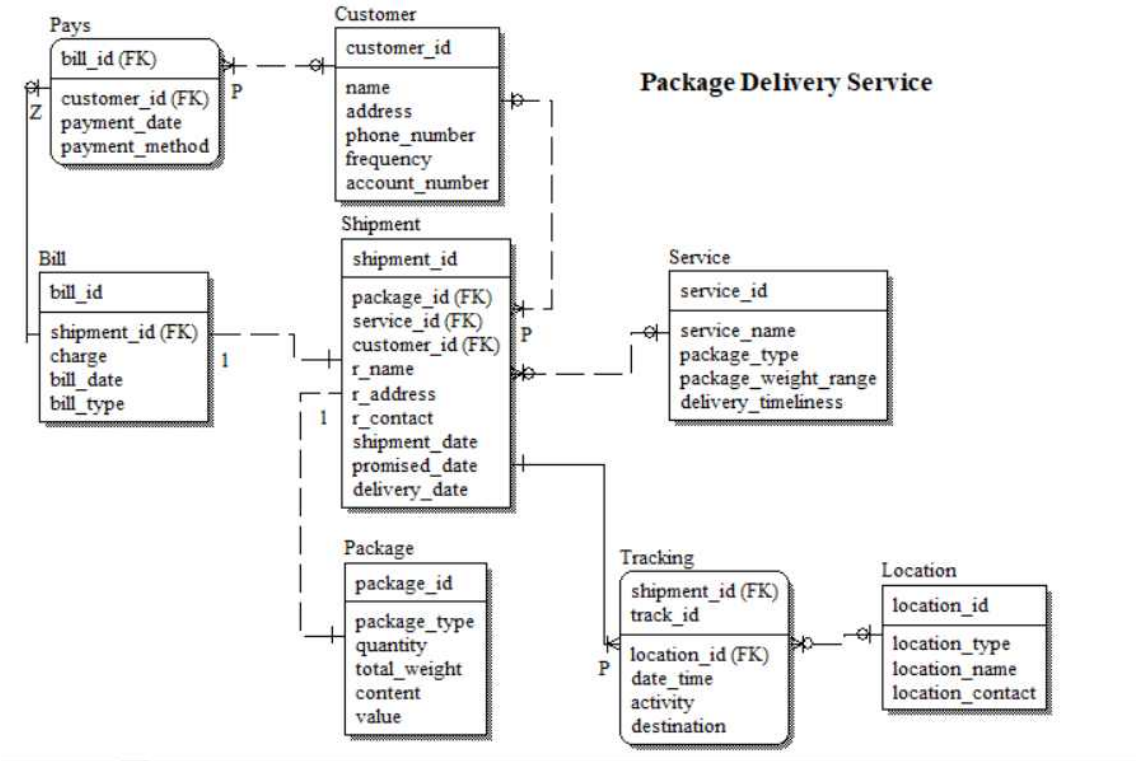
*tracks*

Tracking과 Location의 관계에 해당합니다. 각 Tracking은 하나의 Location만 갖게 되지만, Location은 여러 개의 Tracking에서 나타날 수 있습니다. 따라서 Tracking ->Location은 다대일 관계로 설정했습니다.

### III) 명세서의 Query 처리, How ?

디자인한 DB에 대한 추가적인 설명을 드리기 위해, 요청하신 Query 중 'Find the customer who has shipped the most packages in the past year' 에 대한 정보를 어떻게 찾는지 설명하겠습니다.

[Schema Diagram]




디자인한 DB의 Shipment 테이블은 schema diagram에서 각각 customer_id와 package_id를 갖게 됩니다. 따라서, 우선 shipment에 몇 개의 물품이 담겼는지 package_id와 연동하여 확인할 수 있고, 이를 customer를 기준으로 묶어 각 customer가 shipment에서 총 몇 개의 물품을 부쳤는지 확인할 수 있습니다.

더 자세한 문의사항이 있으신 경우, joobe24@sogang.ac.kr 로 문의주시기 바랍니다 :)