Database System Project 1

담당 교수 : 정성원

이름 : 박준혁

학번 : 20171643

1. 프로젝트 개요

가상의 전자제품 거래 회사 'DBA(Database Administrator)'가 되어, Manager의 요구사항을 만족시키는 데이터베이스를 설계하는 것이 본 프로젝트의 핵심이다. 이번 1차 프로젝트에서는 데이터베이스의 초기 디자인을 구축한다. E-R Model을 그리고, 이를 토대로 Relational Schema까지 만들어본다.

참고로, 전자제품 거래 회사란, 프로젝트 설명에 언급되고 있는 'Best Buy' 웹사이트에 들어가보면 무엇을 이야기하는지 정확히 알 수 있다. 가장 친숙한 예시로는 국내의 'Hi-Mart'도 있다. 다양한 회사의 다양한 전자제품을 취급하고, 고객에게 이들을 중개하고 판매하는, 그러한 회사를 떠올리면 된다.

2. E-R Model

A. Requirements 분석

1. Product

Product는 다양한 카테고리로 분류할 수 있다는 점이 핵심이다. Type, 제조사, 패키지 등이 예시로 언급되고 있다. 따라서, 본 E-R Model에는 Product Entity Set이 있어야 할 것이며, 그 Attribute로 Type, Manufacturer, Package 등을 생각해볼 수 있다.

2. Customer

Manager의 요구사항 설명에 따르면, Customer는 크게 두 가지로 분류된다. 하나는 'Contract Customer', 즉, 회사와 계약이 되어있는 고객이다. 이들은 특정한 계정(Account)을 가지고 있고, 이를 통해 월마다 구매 내역에 대해 일정 금액을 지불한다. 할부를 떠올리면 된다.

또 다른 유형의 고객은 'Infrequent Customer'이다. 직역하면, '드문 드문 방문하는 고객'인데, 본 프로젝트의 맥락에서는 '계약을 하지 않은 고객'이라고 보면 된다. 이들은 카드를 통해 결제하며, Manager에 따르면, 비-온라인 고객의 경우 카드 정보를 굳이 저장할 필요가 없다고 언급하고 있다.

이를 통해 초안으로 생각해볼 수 있는 구조는, Customer라는 Parent Entity

Set이 있고, 여기서 Specification으로서 만들어지는 Contract Customer & Infrequent Customer Entity Set이 있는 형태를 떠올려볼 수 있다.

3. Online Sales

온라인으로 구매한 경우, 해당 상품은 택배나 해운 등의 운송 수단을 통해 전달된다. 우리 회사는 해당 운송 건에 대해 추적 번호를 두어 상태를 확인하고자 한다. Sales에 대한 Entity Set을 두고, 거기서 위의 경우와 유사하게, Inheritance 관계를 만들어 Online Sales와 In-Store Sales를 구분해 취급해볼 수 있겠다.

4. Inventory

한편, 상품의 재고 관리에 대해서도 설명이 있다. 각 상품은 여러 매장과 창고에 분산되어 보관된다. 창고는 매장 재고를 보충하거나, 온라인 구매 건에 대한 분출을 담당한다.

특정 상품의 재고가 일정 Threshold 이하의 개수를 가질 경우 Reorder를 해야 한다. 이때, Reorder는 DB에 그 기록을 남겨야 한다. Reorder 상품이 보충될 경우, 해당 Reorder 건에 대한 기록을 Update하기도 해야 한다.

이러한 설명을 토대로 떠올려볼 수 있는 가장 간단한 아이디어는 다음과 같다.

- 우선, Store와 Warehouse를 나타낼 수 있는 Entity Sets가 필요해 보인다. 이때, Store와 Warehouse를 나눌지 말지가 약간 고민이 될 수 있는데, Warehouse에는 'Store 보충 및 온라인 구매 분출', Store에는 '현장 구매 분출'이라는 확실한 목표가 나뉘어 있으므로, 따로 나누는 것이 합리적이라 판단된다.
- 그러한 Store, Warehouse와, Product가 일련의 Relationship Sets를 구성해야 할 것은 명확하다. 아마, 문맥상 '재고'라는 의미를 지닌 Relationship Set이면 좋으리라.
- 한편, Warehouse의 경우, Product의 제조사와 따로 Relationship Set을 가져야 할 것으로 생각된다. Reorder 내역이 DB에 기록되

어야 하기 때문이다. 따라서, Product와 연관된 또 다른 Entity Set으로 Manufacturer을 두자. 그리고 이 Entity Set이 Warehouse와 Reorder 관계를 가지면 되겠다.

5. Sales Data

판매 정보도 따로 관리해야 한다. 마케팅 목적으로 말이다. 설명서를 보면, 구매 시기, 구매 제품, 제품 그룹, Season, 지역(현장 구매 한정) 등이 기준이 될 것이라 한다. 따라서, 앞서 언급한 Sales Entity Set에 관련된 Attribute들을 추가해야 한다. 또한, Product Entity Set, Store Entity Set과의 연계도 필요할 것으로 예상된다.

B. Example Query 분석

✓ 첫 번째 쿼리(USPS, Tracking number 관련)

앞서 언급한 것처럼, Online Sales Entity Set에는 확실히 Shipper와 Track Number 정보에 대한 Attributes가 있어야 함을 알 수 있다. 운송 중인 상품에 외부 요인으로 인한 손상이 발생했을 시, 이를 표시하기 위한 특정한 Flag 변수도 두면 좋겠다.

한편, In-Store Sales를 고려해 Sales라는 Superclass를 두고, Online Sales와 In-Store Sales가 상속관계를 가지게 할 것이라 했는데, 이때, Generalization에 해당하는 Sales Entity Set은 Customer와도 연계되어야 한다. 손상 건에 대해, 구매자에게 연락을 취한다고 하므로, Customer와 Sales 간에 관계가 정의되어야 한다. Customer에는 연락처 정보가 있어야겠다.

또한, 손상 건에 대해, 구매한 상품들을 조회해, 다시 보내야 한다. 따라서, Sales로부터 Product를 알 수 있도록, Sales와 Product도 관계를 가져야 한다. 새로운 Shipment를 만드는 것은, MySQL, Embedded SQL 등으로 DBA가 처리해야 할 부분으로 보인다.

✓ **두 번째 쿼리(최대 구매자 관련)**

우선, Sales에 '구매 일자' 정보를 추가해야 함은 명확해 보인다. 그리고, 이 때, 특정 ID의 고객에 대해, 해당 고객이 구매한 모든 Sales 정보를 찾아, 이들의 구매 가격 합을 알아야 한다. 따라서, Sales와 Product와 관계를 가질 때, 가격 정보도 있어야 할 것이다. 가격은 기본적으로 Product 쪽에 있으면 좋을 것이다. Sales에도 물론 가격 정보를 담을 수 있지만, 정보의 중복(Data Redundancy)이 발생할 가능성이 높아지므로, 굳이 그러진 않겠다.

✓ **세 번째, 네 번째 쿼리(최대 판매 상품 관련)**

세 번째 쿼리의 경우, 두 번째 쿼리 분석 내용과 관련이 깊다. Sales와 Product의 Relationship에 의해 자연스럽게 커버할 수 있다.

네 번째 쿼리는 상당히 중요하다. Unit Sale이란 것은, Product 단위 판매를 의미한다. 이는 곧, 본 프로젝트 문제 상황은 Package 판매를 늘 고려해야 함을 뜻한다. 즉, Package에 대한 처리가 필요하다. 이를 어떻게 처리할 수 있을까 장시간의 고민을 거친 후, 다음과 같은 아이디어를 떠올렸다.

"Product와 'Market Item(판매 상품)'을 구분하자. 그리고, Market Item과 Product가 package라는 'Many-to-Many' 관계를 가지게 하자. 그렇게 하면, Package도 처리할 수 있고, 동시에 앞서 계속 언급했던 Product와 Sales의 관계성도 계속 유지할 수 있다."

따라서, Market Item이라는 새로운 Entity Set을 도입한다. 이전까지 언급했던 Product의 역할은 모두 이 Market Item이 대체한다. Product는 정말 말 그대로 Product 그 자체로서의 역할만 수행한다. 이 Product가 단독으로, 또는 복수로 Package가 되어 Market Item이 되고, 그 Market Item이 Sales의 대상이 되는 것이다. 이를 위해, Market Item Entity Set에는 Price를 명시할 Attribute가 있어야겠다.

✓ **다섯 번째 쿼리(California Store 관련)**

이 쿼리를 보면, 앞서 언급한 Store와 Product 간의 Relationship Set이 필요함은 당연지사고, 그 의미를 'Stock in store'로 두면 좋겠다는 것을 알 수 있다. Product와 Store가 관계를 가지는데, 상품 종류와 매장의 조합이 모두 각각 Unique한 개체이므로, 'Many-to-Many' 관계이면 적당할 것으로 보인다.

한편, Store에는 지역 정보가 있어야 한다. 본인은 이를 Region이라는 Attribute를 도입해 처리하겠다.

✓ **여섯 번째 쿼리(운송 예정 시간 관련)**

상기한 Online Sales Entity Set에는 Promised Date 정보를 담는 Attribute도 필요해 보인다. SQL을 이용해 현재 날짜와 이 Promised Date 정보를 비교해서 상태를 알 수 있으리라. 다만, '미리 도착하는 경우'도 있으므로, 보다 확실하게 Flag 변수를 두어 처리하는 것이 정확해 보인다. 해당 Flag는 '도착', '미-도착'을 나타내고, 만일, Promised Date가 지났는데 Flag가 '미-도착'이면, 회사가 이를 확인해 처리하면 될 것이다.

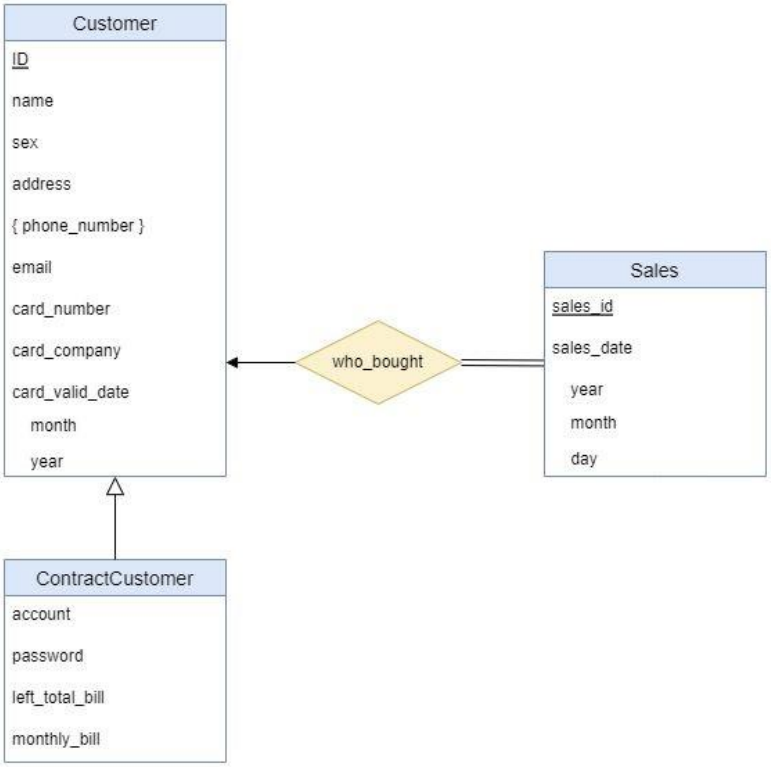
✓ **일곱 번째 쿼리(Monthly Bill 관련)**

Contract Customer에 대해 Bill과 관련된 Attributes를 추가해야 한다. 다양한 방법이 가능해 보이는데, 본인은 '남은 총액(Left Total Bill)'과, '지난 달 기준 총액에 대한 Monthly Bill'을 두어 처리하겠다. Monthly Bill의 계산은, 각 계약 회원들의 ID에 대해, 매칭하는 Sales 정보들(기간을 지난 달로 한정)을 모두 찾아, 총액에 더해주고, 그 총액을 분할한 금액을 계산해서 수행할 수 있을 것이다.

C. E-R Model 구축

지금까지 언급한 내용들을 토대로, E-R 모델을 구축한다. 각 부분 별로 쪼개서 설명을 진행하겠다.

(1) Customer 부분



여기서 가장 핵심은, 본인은 Online Customer와 In-Store Customer를 따로 구분하지 않았다는 것이다. Manager는 Computer Science, Database적 지식이 없는 사람이라서, 그 둘을 구분하는 지시를 했다고 판단했다.

"굳이 Online Customer와 In-Store Customer를 DB 관점에서 구분할 필요가 있는가? In-Store Customer의 경우, 필요한 정보만 남겨두고 모두 Null로 처리해서 Tuple을 기입해놓으면 되지 않는가."

물론, DB 관리 측면에서 Null의 존재는 좋지 않다고 했다. 하지만, In-Store 고객의 경우, 사실상 Sales 기록을 남긴다는 점 외에는 DB 관리 측면에서 굳이 필요가 없다. 즉, 구분을 위한 ID 외의 Attribute를 사용할 상황이 없는 것이다. 물론, 상기한 쿼리 예시에서, 최다 구매 고객을 찾을 때, 우연으로 비회원 매장 고객이 선정될 가능성도 없는 것은 아니지만, 이 역시, Null값이 있는 것이 전혀 문제가 되지 않는다. 본 프로젝트 상황에서, Customer를 온라

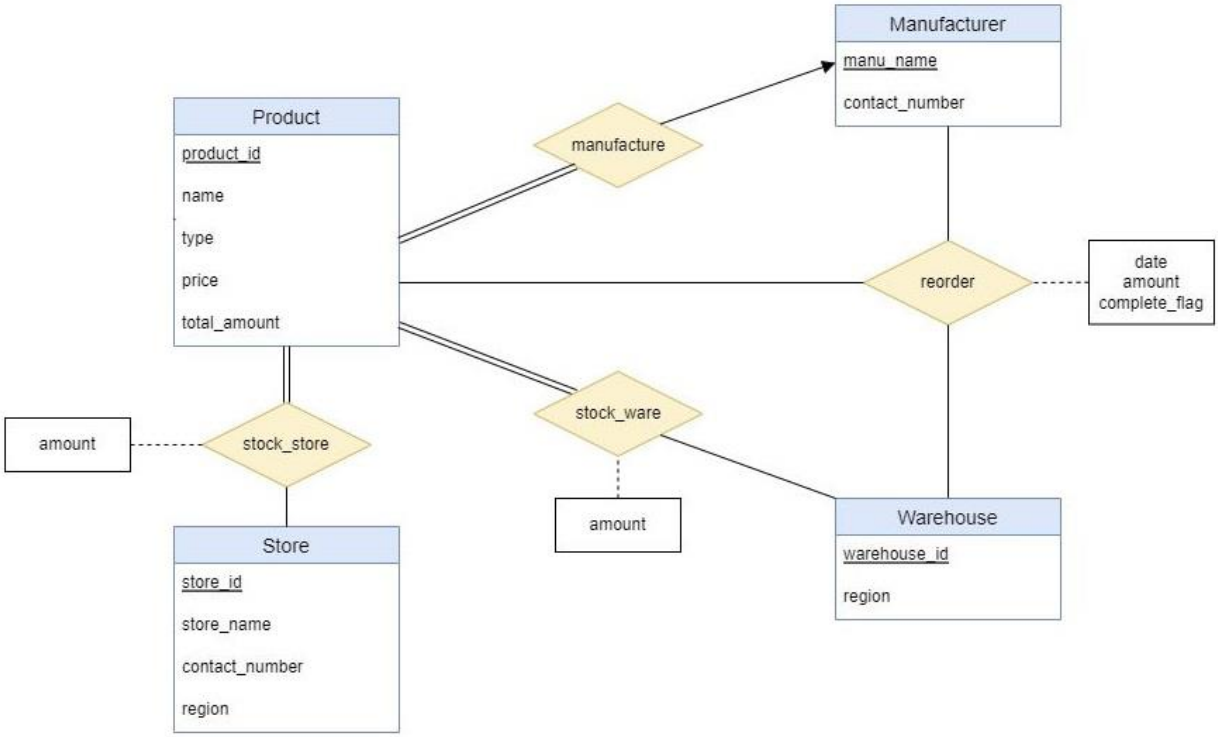
인과 비-온라인으로 구분하지 않는 것이 가져다 주는 불편함은 사실상 없다고 보면 된다. 그냥, Card 정보 관련 속성에 Null이 있으면, 비회원 정보라고 보면 되는 것이다.

한편, Contract Customer는 이러한 Customer를 상속받는다. 앞서 언급한 각 종 속성들이 추가되었음을 알 수 있다.

이때, Sales에선 Customer 정보를 필요로 한다. 필요로 하되, Sales 내에서 Customer의 ID는 굳이 Unique할 필요는 없다. 즉, Sales ID가 있는 이상, Sales의 구분 기준으로 굳이 고객 ID를 사용할 필요가 없다. 따라서, 위와 같이 'Many 쪽이 Total Participation인 One-to-Many' Relationship Set 'who_bought'을 정의한다.

Sales의 경우, 구매 날짜를 기입하는 정보를 담고 있음을 주목하자. sales_id는 Sales를 구분하는 Primary Key로, 각 구매마다 Increment되는 정수로 표현하면 되겠다(다양한 방법이 가능). Customer의 경우, 고객 성함, 성별, 연락처(Multivalued), E-mail, 카드 정보 등을 담고 있음도 주목하자.

(2) Product과 Store, Warehouse, Manufacturer 부분



Product는 Primary Key로 고유한 ID를 가진다. 제품명, 제품 유형, 가격, 재고 총량 등의 속성을 가진다. 그리고 이러한 Product는 Manufacturer와 관계를 가진다. 제조사를 나타내는 Manufacturer는 회사명을 Primary Key로 한다. 연락처 정보도 가진다. 각 제품은 제조사를 일반 속성으로 가지면 되므로, 역시나 위와 마찬가지로 'Many-to-One with Total Participation of Many Side'의 관계(manufacture)를 만들어주면 되겠다. 다만, 제조사명이 Null이 되면 안되므로, 실제 구현 시에는 Not Null 조건을 걸어주는 것이 좋겠다. 이는 상기한 Sales와 Customer 관계에서의 ID 속성(Sales의 Foreign Key)도 마찬가지이다.

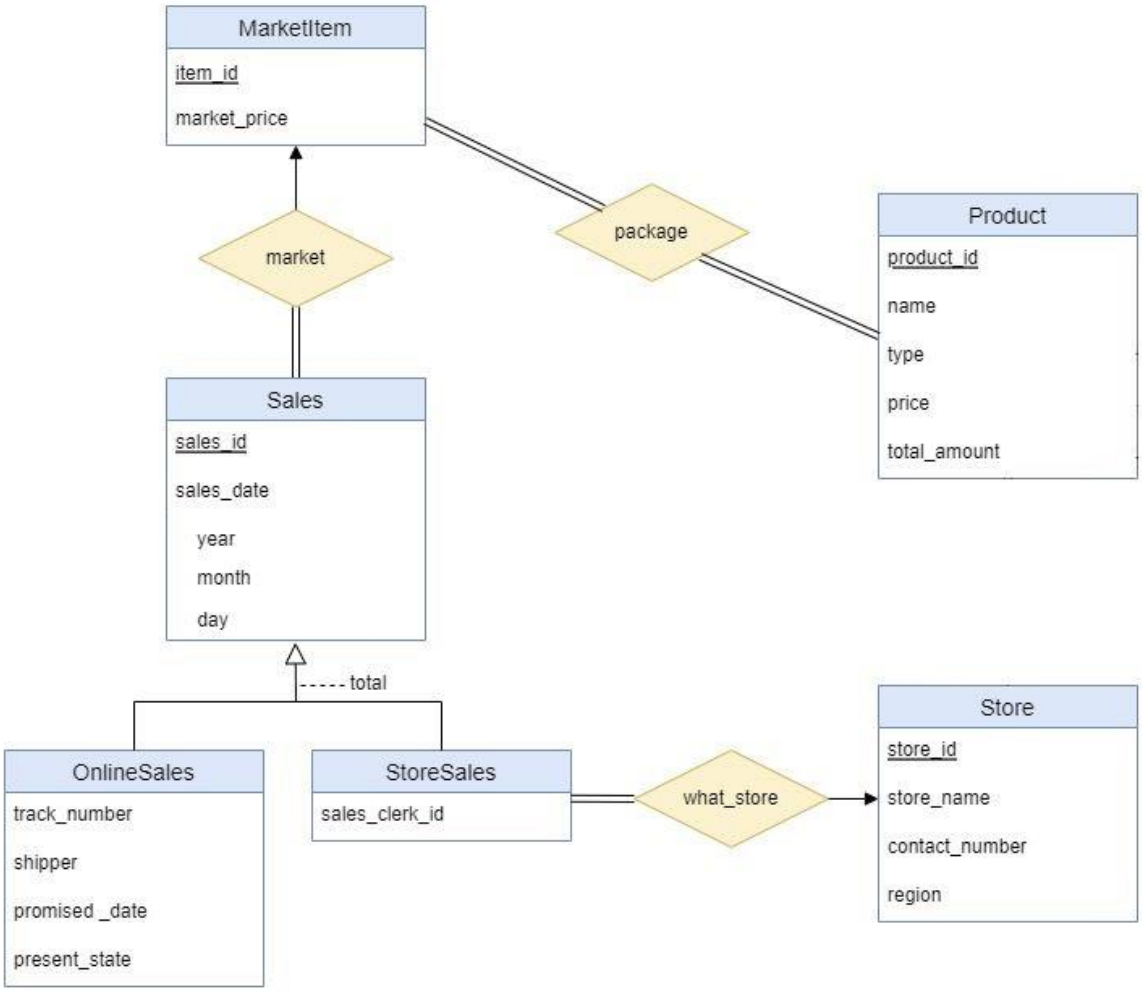
그리고 Product는 Store, Warehouse와 각각 '재고' 관계(stock_store, stock_ware)를 가진다. 이들은 (상품명, 매장/창고) 조합이 모두 Unique해야하므로 'Many-to-Many' 관계로 정립하는 것이 바람직하다. 이때, 두 Relationship Set 모두 amount라는 별도의 Attribute를 가져야 한다. Product의 total_amount 속성은, 각 창고와 매장에 있는 해당 제품의 재고 총량을 나타낸다. 따라서, 실제 구현 시 우리는 Assertion과 같은 수단을 이용해 '실제 재고 총량'과 total_amount 값이 일치하도록 조건을 만들어주어야 할 것이다.

한편, Product, Warehouse, Manufacturer는 reorder라는 Ternary 관계를 갖는다. (상품명, 창고, 회사)의 조합이 Unique하므로, 모든 Side는 Many로 구성해 reorder의 Primary Key가 세 Entity Set의 Primary Key가 되도록 한다. reorder가 존재하는 이유는, 앞서 언급한 것처럼, DB에 재-주문 정보를 기록하고, 추후 재-주문 완료 시 '완료 표시'를 하기 위함이다.

이외에, region, contact_number와 같은 추가 정보들도 넣어주었다.

(3) Sales, Market Item 부분

(다음 장)



앞서 언급한 Market Item Entity Set이 있다. Product와는 'Many-to-Many' 관계(package)를 가지도록 설계했다. Package마다 복수의 Product가 들어갈 수 있으므로, package라는 관계에서 이 Matching 관계를 기록해두어, Market Item으로 간접 참조하도록 말이다. 이때, Product나 Market Item이나, 반드시 이 package 관계에 참여해야 하므로, 양 쪽 모두 Total Participation으로 구성해주었음도 주목하자.

이러한 Market Item은, Sales가 Customer와 가지는 관계와 같은 관계를 가진다. MarketItem의 Primary Key인 item_id가 Sales 내에서 일반 속성으로 존재하면 되기 때문이다. 물론, Not Null 조건은 필요할 것이다.

여담으로, 이 Sales ~ Market Item, Sales ~ Customer 관계 정립에 상당한 시간을 투자하였음을 밝힌다. 초기에는, Sales라는 정보는 '의미'만을 놓고 보았을 땐, 결국 Market Item과 Customer가 모두 존재해야 존재할 수 있으므로 Sales를 Market Item과 Customer를 Identifying Entity Set으로 하는 Weak Entity Set이라 생각하였다. 하지만, 이렇게 하면 다음과 같은 문제들이 발생했다.

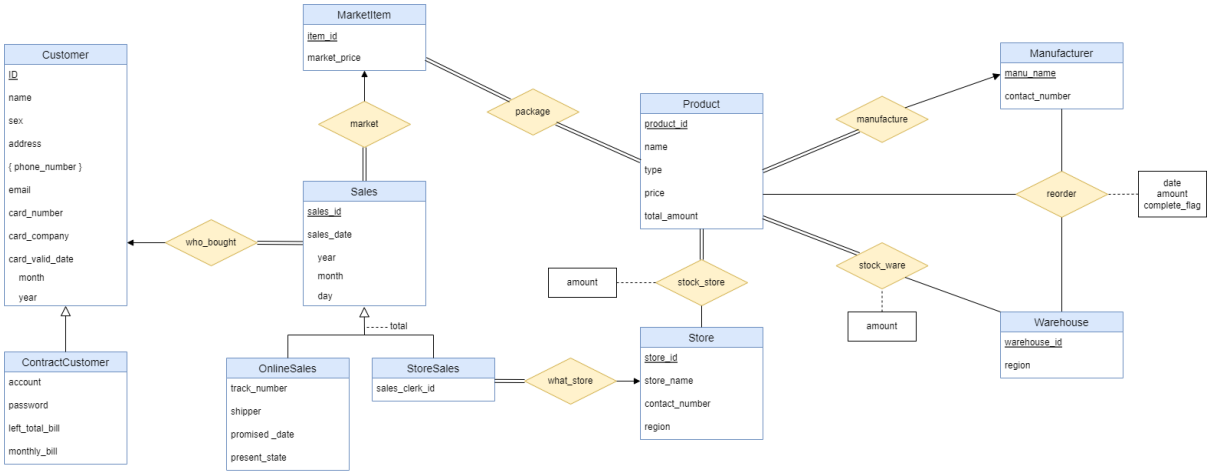
- (sales_id, ID, item_id)의 조합이 모두 Unique할 경우, 같은 sales_id에 대한 잘못된 정보나 Inconsistency가 발생할 가능성이 높아진다.
- 또한, 본 수업 교재 외의 다른 Database 교재에 따르면, Weak Entity Set은 일반적으로 Identifying Entity Set을 하나만 가진다는 서술이 존재한다.
- 그리고 무엇보다, ID와 item_id는 어차피 Sales 내에서 중복이 가능하므로, 그냥 일반 Attribute로 두어도 전혀 문제가 없다. Not Null 조건만 있다면 말이다.

따라서, 'Many-to-One with Total Participation of Many Side'로 Cover한다.

한편, Sales는, Online Sales와 Store Sales로 Disjoint한 Inheritance 관계를 가지도록 설계했다. Sales는 반드시 두 Entity Set 중 하나로 분류될 수 있기 때문에 Disjoint Specification이라 보는 것이 합당하기 때문이다. 당연히 Total이다.

Online Sales와 Store Sales에 적당히 필요한 속성을 추가해주었다. 이때, Store Sales의 경우, Store 정보도 필요하므로, 위와 마찬가지로 'Many-to-One with Total Participation of Many Side' 관계(what_store)를 만들어준다.

이러한 과정을 거쳐서 최종적으로 아래와 같은 E-R Diagram을 만들 수 있다. 동봉된(첨부된) 'E-R diagram'을 통해 자세하게 확인할 수 있다.



3. Relational Schema Diagram

위의 과정을 통해 제작한 'E-R Diagram'을 Relational Schema Diagram으로 Reduction을 수행하는 과정은 어렵지 않다. 강의에서 배운 내용을 그대로 적용하면 된다.

1. Inheritance 관계

상속 관계 표현 방법은 강의에서 배운 것처럼, 크게 두 가지 방식이 있는데, 본인은 Primary Key만을 상속하는 방식으로 구축했다. Cardinality가 One-to-One/Zero인 Identifying Relationship을 만들어 연결하면 된다. 당연히, 상위 Relation의 Primary Key가 하위 Relation의 Primary Key이자, Foreign Key로 작동하게 된다.

2. Multivalued 처리

Customer Entity Set에 있는 phone_number라는 Attribute는 Multivalued이다. 이는 강의에서 다룬 예시와 정확히 같은 상황으로, 별도의 Schema로 분리해서 ID를 이용해 연결해주면 된다. 'One-to-Many'의 Cardinality를 Mapping 하자.

3. One-to-Many with Total Participation of Many Side 관계

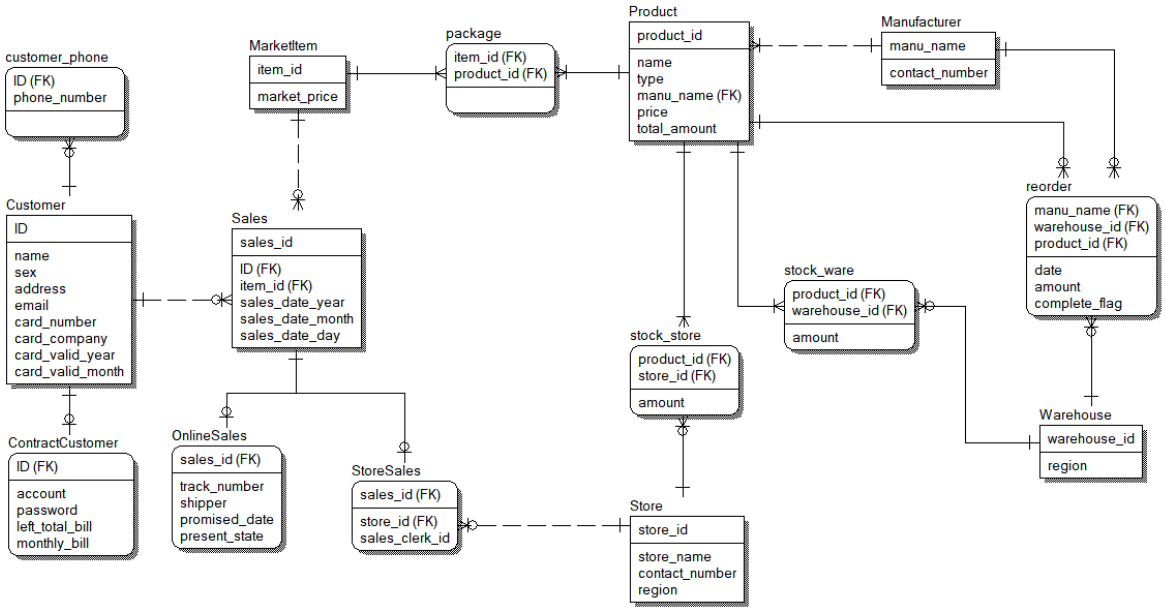
위 E-R Diagram에서 가장 자주 등장하는 관계 유형이다. 이 역시 마찬가지로 강

의에서 배운 것처럼, One 쪽에 해당하는 Entity Set의 Primary Key를 Many 쪽에 일반 Attribute로 포함해주면 된다. 별도의 Relation을 추가로 만들지는 않고 말이다. 위의 E-R Diagram에서, 'Sales ~ Market Item', 'StoreSales ~ Store', 'Product ~ Manufacturer' 관계들이 이에 해당한다.

4. Many-to-Many 관계

가장 간단하다. 해당 Relationship Set의 이름과 같은 이름의 Relation을 새로 만들고, 관계된 Entity Set의 Primary Key들을 모두 가져와 Primary Key로 취급하면 된다. Ternary Relationship Set도 마찬가지로 처리한다. 위의 E-R Diagram에선, 'MarketItem ~ Product', 'Product ~ Manufacturer ~ Warehouse', 'Product ~ Store', 'Product ~ Warehouse'가 바로 그러하다.

이러한 Routine을 이용해 어렵지 않게 Reduction을 수행할 수 있다. 결과는 아래와 같다. (동봉/첨부된 ERwin 파일로 확인할 수 있다)



이때, 'One-to-Many with Total Participation of Many Side' 유형만 제외하곤, Total Participation이 있는 경우, Reduction 시 Cardinality 하한을 1로 설정하는 것도 중요하다. Manufacturer와 Product의 관계는 예외인데, 모든 제품은 반드시 제조사를 가지고, 동시에 '기록된 제조사'들 역시 마찬가지로 제품이 있기 때문에 '기록'된 것이므로, 이들의 하한은 'One-to-Many with Total Participation of Many Side'임에도 1로 두었다. 나머지

'One-to-Many with Total Participation of Many Side'의 경우, Many 쪽의 하한은 0이다. 의미를 고려했을 때, 이렇게 구성하는 것이 합당하기 때문이다.

예를 들어, 'MarketItem과 Sales'의 관계를 보면, 하나의 Market Item은 복수의 Sales와 맵핑되지만, 동시에 맵핑이 안될 수도 있다. 따라서, 하한은 0이 합당하다.

이렇게 해서, 첫 번째 프로젝트를 마무리한다. 최종적으로 직접 예시 Query를 작성해 가며 본 구조들을 검증해보았기 때문에 전반적으로 논리적 비약이 많지 않을 것이라 생각한다. 다만, 잦은 참조 관계가 필요하기 때문에, 수행 속도와 효율 측면에서는 그다지 우수하지 않을 수 있다고 생각한다. 이와 관련해서는, 이어지는 프로젝트를 수행하면서 추가 검증 및 보완을 하도록 하겠다.