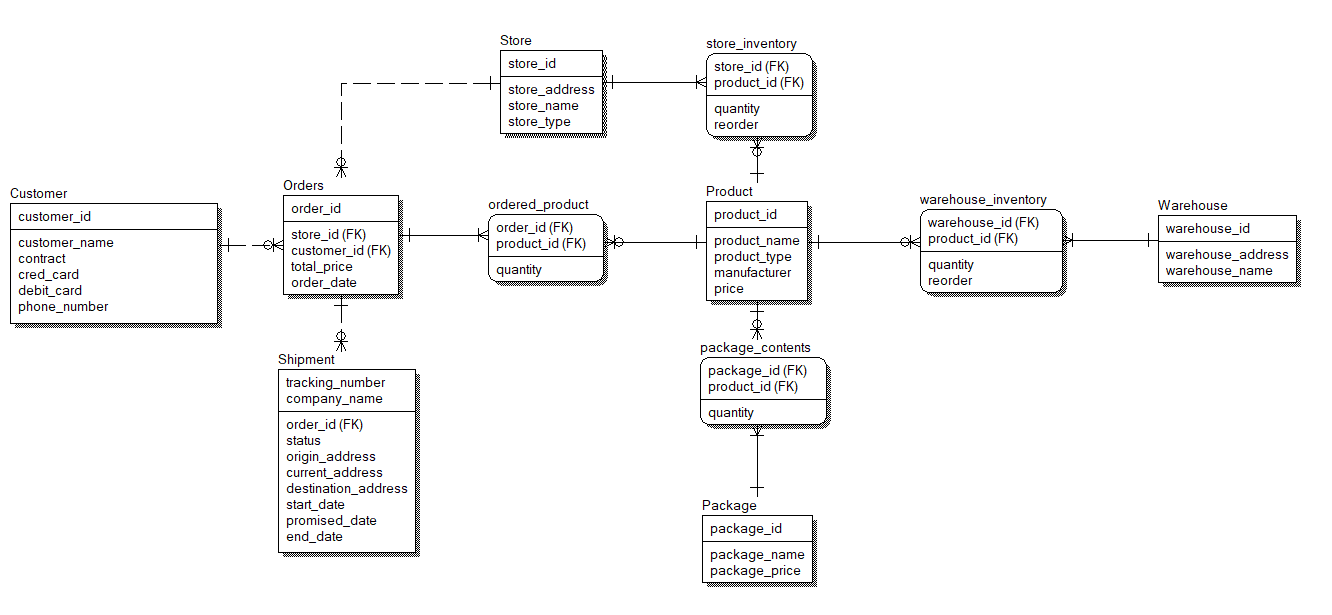
**CSE4110 – Database System**

**Project 2. Normalization and Query Processing**

**20171645 박찬우**

1. **Decomposition to BCNF**



우선, 후에 서술할 Query를 고려하여 기존의 logical schema에서 몇가지 변경점이 추가되었다. Customer Table에 고객의 연락처를 저장하는 phone\_number attribute를 추가하였고, package\_contents 에 특정 package에 어떤 종류의 물건이 몇 개 들어있는지를 나타내는 quantity attribute를 추가하였다. 추가로, 기존의 Order table의 이름을 Orders로 변경했는데, 이는 query중 order가 query 명령어중 하나여서 table명으로 인식이 안되기 때문이다.

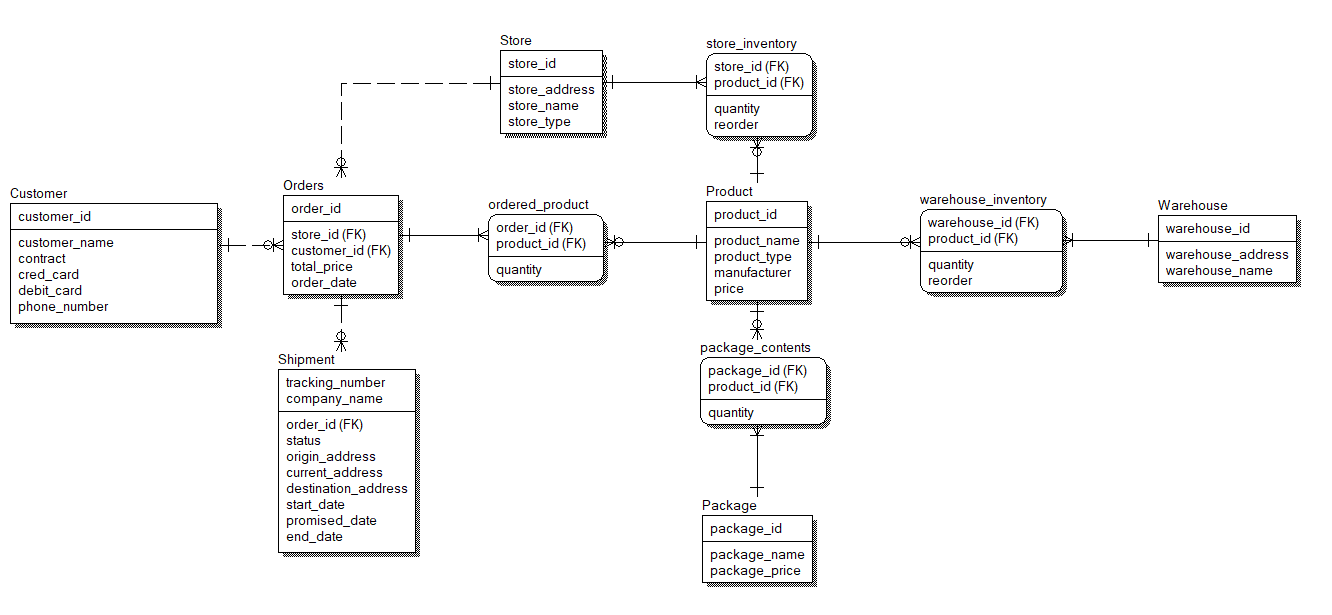
위 내용은 모두 기존 Schema에서 Query를 위해 필요한 정보를 추가한 것들이고, 기존 Relational Schema에서 BCNF form으로 decompose 해야 하는데, 기존 Relational Schema의 모든 Table이 BCNF form이므로, 추가적인 decomposition 과정은 없다.

기존의 Relational Schema의 Functional dependency는 모두 a->b 에 대하여 a가 Relational Schema의 super key인 경우, 예를들어 [ customer\_id -> customer\_name, contract, cred\_card, debit\_card, phone\_number ] 인 경우와 trivial 한 경우, 예를 들어 [ customer\_name , contract ] -> [ customer\_name, contract ] 와 같은 두 가지 종류를 제외한 functional dependency가 존재하지 않는다. 따라서, 전부 BCNF form 형태를 취하고 있으므로, 별다른 decomposition 과정이 없다.

간단히 Customer relation에 대해 BCNF simplified test를 해보자면, 현재 Customer에 존재하는 functional dependency는 trivial한 functional dependency를 제외하면 [ customer\_id -> customer\_name, contract, cred\_card, debit\_card, phone\_number ] 하나이다. 동명이인이 있을 수 있고, contract 여부는 binary하며 cred\_card, debit\_card, phone\_number는 nullable하므로 해당 attribute로 functional dependency를 형성할 수 없다. 따라서 trivial 한 functional dependency를 제외한 위의 하나의 functional dependency의 경우, a->b 형태에서 a가 relation의 primary key 이므로, super key이다. 따라서, Customer는 BCNF form을 만족한다.

다른 모든 Relation 역시 비슷하게 trivial 및 a가 super key인 functional dependency를 제외한 다른 functional dependency가 없으므로, 전부 BCNF form을 만족한다.

1. **Physical Schema**



각 Entity의 attribute에 대해 먼저 살펴본다.

텍스트이(가) 표시된 사진

자동 생성된 설명

customer\_id 는 각 고객을 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

contract는 고객이 회사와 계약을 했는지 여부를 나타내는 attribute로, boolean type의 데이터를 가지며 Null이 허용되지 않는다. contraint는 customer\_contract로, boolean data인 yes와 no 값만을 가질수 있도록 제약한다.

cred\_card는 고객의 신용카드 정보를 저장하는 attribute로, CHAR(19) type 데이터를 가지며 Null이 허용된다. card\_form 이라는 contraint로 0000-0000-0000-0000 형태를 가진 값만 입력받을 수 있도록 제약한다.

debit\_card는 고객의 체크카드 정보를 저장하는 attribute로, 이름을 제외하면 cred\_card와 동일하다.

phone\_number는 고객의 전화번호를 저장하는 attribute로, CHAR(13) type 데이터를 가지며 Null이 허용된다. phone\_form 이라는 contraint로 000-0000-0000 형태의 값을 입력받을 수 있도록 제약한다.

이 Entity의 primary key는 customer\_id 이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

customer\_id 는 각 주문을 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

store\_id 와 customer\_id는 각각 Store 와 Customer로부터 받아온 foreign key로, 둘 모두 INT type의 데이터를 가지며 Null이 허용되지 않는다.

total\_price는 해당 주문의 총 금액($)을 저장하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

order\_date는 주문한 날짜 정보를 저장하는 attribute로, DATETIME type의 데이터를 가지며 Null이 허용되지 않는다. DATETIME type의 도메인은 Datetime으로, “2022-06-01 12:00:00” 와 같은 형태의 데이터를 저장할 수 있다.

해당 Entity의 primary key는 order\_id이고, foreign key는 store\_id, customer\_id 이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

tracking\_number는 송장번호, 즉 배송 정보의 id 역할을 하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

company\_name은 배송하는 회사의 이름을 저장하는 attribute로, VARCHAR(100) type 데이터를 가져 최대 100자까지 문자열을 저장할 수 있고, Null이 허용되지 않는다.

order\_id는 Orders로부터 받아온 foreign key로, INT type 데이터를 가지며 null이 허용되지 않는다.

status는 현재 배송이 어떤 상태인지를 나타내는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다. Shipment\_Status 라는 constraint에 의해 “Information” “At Pickup” “Arrived at Carrier” “In transmit” “Out for Delivery” “Delivered” “Stopped by accident”값들중 하나만 입력받을 수 있다. 해당 값들은 각각 ‘배송정보 접수’, ‘화물 접수’, ‘배송사 도착’, ‘배송중’, ‘배송직전’, ‘배송완료’, ‘문제 발생으로 배송중단’ 을 의미한다.

origin\_address, current\_address, destination\_address는 각각 배송 출발지, 현재 위치, 배송 목적지에 대한 정보를 저장하는 attribute로 모두 VARCHAR(100) type 데이터를 저장하며 Null이 허용되지 않는다.

start\_date, promised\_date, end\_date는 각각 배송 시작 시간, 예정 도착 시간, 도착 시간에 대한 정보를 저장하는 attribute로 모두 DATETIME type 데이터를 저장하며, end\_date만 Null이 허용되고, 나머지는 Null이 허용되지 않는다.

이 Entity의 primary key는 tracking\_number와 company\_name이고, foreign key는 order\_id이다.

테이블이(가) 표시된 사진

자동 생성된 설명

store\_id 는 각 상점을 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

store\_address는 상점의 주소를 저장하는 attribute로, VARCHAR(100) type의 데이터를 가지며 Null이 허용되지 않는다.

store\_name은 상점의 이름을 저장하는 attribute로, VARCHAR(50) type의 데이터를 가지며 Null이 허용되지 않는다.

store\_type은 상점이 온라인 상점인지 오프라인 상점인지를 나타내는 attribute로, VARCHAR(7) type 데이터를 저장하며 Null이 허용되지 않는다. Store\_type constraint에 의해 online, offline 두가지 값만을 가질 수 있도록 제약된다.

이 Entity의 primary key는 store\_id이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

warehouse\_id 는 각 창고를 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

warehouse\_address는 창고의 주소를 저장하는 attribute로, VARCHAR(100) type의 데이터를 가지며 Null이 허용되지 않는다.

warehouse\_name은 창고의 이름을 저장하는 attribute로, VARCHAR(50) type의 데이터를 가지며 Null이 허용되지 않는다.

이 Entity의 primary key는 warehouse\_id이다.

텍스트이(가) 표시된 사진

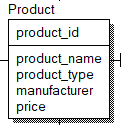
자동 생성된 설명

package\_id 는 여러 상품의 묶음인 package를 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

package\_name은 묶음의 이름을 나타내는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다.

package\_price는 묶음의 가격을 타나내는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

이 Entity의 primary key는 package\_id이다.



product\_id 는 상품을 구분하는 id 역할을 하는 attribute로, INT type의 데이터를 가지며 Null이 허용되지 않는다.

product\_name은 상품의 이름을 저장하는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다.

product\_type은 상품의 종류를 저장하는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다.

manufacturer는 상품의 제조사를 저장하는 attribute로, VARCHAR(50) type 데이터를 가지며 Null이 허용되지 않는다.

price는 상품의 가격을 저장하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

이 Entity의 primary key는 product\_id이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

store\_id와 product\_id는 각각 Store와 Product로부터 가져온 foreign key로, 모두 INT type 데이터를 가지며 Null을 허용하지 않는다.

quantity는 상점에 있는 물건의 개수를 저장하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

reorder는 상점의 해당 상품의 재주문에 대한 정보를 저장하는 attribute로, VARCHAR(20) type 데이터를 가지며 Null이 허용되지 않는다. Reorder contraint로 인해 각 “Reorder need” “Reordering” “Not Need” 세가지 값만을 가지도록 제약된다. 각 값은 재주문이 필요함, 재주문중, 재주문이 끝났거나 필요치 않음을 의미한다.

이 Entity의 primary key는 store\_id와 product\_id이며, foreign\_key 역시 store\_id, product\_id이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

warehouse\_id와 product\_id는 각각 Warehouse와 Product로부터 가져온 foreign key로, 모두 INT type 데이터를 가지며 Null을 허용하지 않는다.

quantity는 창고에 있는 물건의 개수를 저장하는 attribute로, INT type 데이터를 가지며 Null이 허용되지 않는다.

reorder는 창고의 해당 상품의 재주문에 대한 정보를 저장하는 attribute로, VARCHAR(20) type 데이터를 가지며 Null이 허용되지 않는다. Reorder contraint로 인해 각 “Reorder need” “Reordering” “Not Need” 세가지 값만을 가지도록 제약된다. 각 값은 재주문이 필요함, 재주문중, 재주문이 끝났거나 필요치 않음을 의미한다.

이 Entity의 primary key는 warehouse\_id와 product\_id이며, foreign\_key 역시 warehouse\_id, product\_id이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

order\_id와 product\_id는 각각 Orders와 Product로부터 가져온 foreign key로, 모두 INT type 데이터를 가지며 Null을 허용하지 않는다.

quantity는 해당 주문에서 어떤 상품을 몇 개 주문했는지를 의미하는 attribute로, INT type 데이터를 가지며 Null을 허용하지 않는다.

이 Entity의 primary key는 order\_id, product\_id이고 foreign key 역시 order\_id, product\_id이다.

텍스트이(가) 표시된 사진

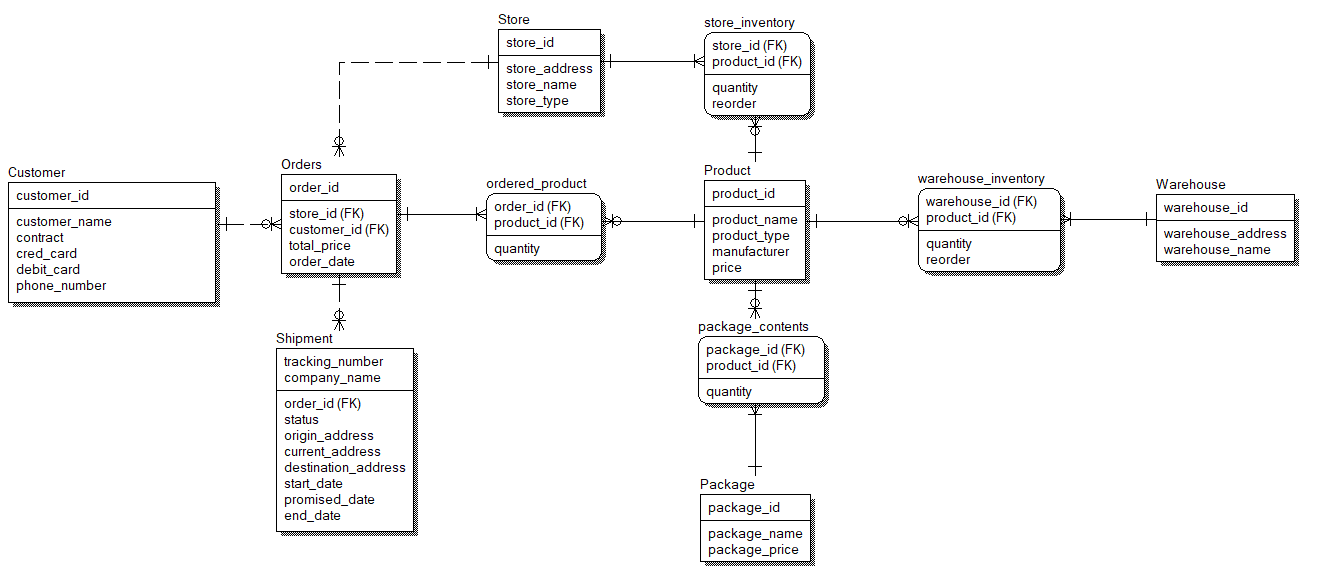
자동 생성된 설명

package\_id와 product\_id는 각각 Package와 Product로부터 가져온 foreign key로, 모두 INT type 데이터를 가지며 Null을 허용하지 않는다.

quantity는 해당 묶음에서 어떤 상품이 몇 개 포함됬는지를 의미하는 attribute로, INT type 데이터를 가지며 Null을 허용하지 않는다.

이 Entity의 primary key는 package\_id, product\_id이고 foreign key 역시 package\_id, product\_id이다.

이제 각 Entity 사이의 관계를 살펴본다.



**Customer와 Orders 에서** order는 total participation해야 하며, Customer와 Orders 사이의 mapping cardinality는 one-to-many이다. 다시말해, customer는 order를 하나도 하지 않을 수 있고, 하나만 할 수 있으며, 여러 order를 할 수도 있다. 따라서, customer – order 사이의 relationship은 one-to-zero-one-or-more 로 설정한다. schema reduction rule에 따라, order는 customer의 primary key를 foreign key로 갖게 된다. 이때 customer\_id 는 order의 primary key로 불필요하므로, 둘 사이의 relationship은 non-identifying relationship이다.

**Orders와 Shipment 에서** shipment가 total participation해야 하며, order와 shipment 사이의 mapping cardinality는 one-to-many 이다. 다시말해, shipment는 하나의 order를 갖고, order는 shipment를 갖지 않을수도, 하나 이상을 가질수도 있다.

즉, order - shipment의 Relation은 one-to-zero-one-or-more 관계로 설정한다. 이에 따라 shipment는 order의 primary key를 foreign key로 갖는데, 그것이 order\_id이다. 이때 order\_id는 shipment의 primary\_key로 불필요하므로 둘 사이의 relationship은 non-identifying relationship이다.

**Orders와 Store** **에서** order는 total participation해야 하며, order와 store 사이의 mapping cardinality는 many-to-one 이다. 즉, 한 매장이 많은 주문을 가질수 있지만, 한 주문이 많은 매장을 가질 순 없다. 한 매장은 주문이 없어도 되므로, store와 order 사이의 관계는 one-to-zero-one-or-more 관계를 이루게 된다. 이에 따라 order는 store의 primary key인 store\_id를 foreign key로 받게 되는데, order의 primary key로 필요하지 않으므로 둘 사이의 관계는 non-identifying relationship이다.

**Orders와 ordered\_product, Product 에서** 한번의 주문은 여러 제품을 포함할 수 있고, 아무 제품도 없을 수는 없으므로, order와 ordered\_product 사이의 relationship은 one-to-one-or-more relationship이 성립한다. 한 제품이 여러 주문에 들어있을 수 있고, 아무 주문에도 안들어있어도 괜찮으므로 product와 ordered\_product 사이의 relationship은 one-to-zero-one-or-many relationship이 성립한다. 두 relation 모두 ordered\_product가 상대방으로부터 빌린 foreign key를 primary key로 가지므로, identifying relation이다.

**Store와 store\_inventory, Product 에서** 한 상점에 여러 제품이 있을 수 있지만, 상점에서 아무 제품도 팔지 않을 수는 없으므로 store와 store\_inventory 사이의 관계는 one-to-one-or-more relationship이 성립한다. 한 제품이 여러 상점에 있을 수 있고, 특정 제품은 아무 상점에서도 팔지 않을 수 있다. 따라서, product와 store\_inventory 사이의 relationship은 one-to-zero-one-or-more relationship이 성립한다. 두 relation 모두 store\_inventory가 상대방으로부터 빌린 foreign key를 primary key로 가지므로, identifying relation이다.

**Warehouse와 warehouse\_inventory, Product 에서** 한 제품이 여러 창고에 있을 수 있고, 아무 창고에 없을수도 있다. 따라서, product와 warehouse\_inventory는 one-to-zero-one-or-many relationship이 성립한다. 하지만, 한 창고에 여러 제품이 있을 수는 있지만, 창고에 아무 제품도 보관되지 않을 수는 없으므로 warehouse와 warehouse\_inventory 사이의 relationship은 one-to-one-or-more relationship이 성립한다. 두 relation 모두 warehouse\_inventory가 상대방으로부터 빌린 foreign key를 primary key로 가지므로, identifying relation이다.

**Package와 package\_contents, Product 에서** 한 제품이 여러 패키지에 들어있을수도, 아무 패키지에 없을수도 있으므로 one-to-zero-one-or-more relationship, 한 패키지에 여러 제품이 있을순 있지만 아무 제품이 없을 수는 없으므로 one-to-one-or-more relationship이 성립한다. 두 relation 모두 package\_contents가 상대방으로부터 빌린 foreign key를 primary key로 가지므로, identifying relation이다.

1. **Table Creation**

위에서 다룬 physical schema를 참조하여 Query에 사용할 데이터베이스를 위한 Table과 각 Table의 Value를 추가한다. 각 Table의 Value는 최소 15개 이상을 집어넣는다.

CRUD query를 위해 text file을 사용한 file input을 사용해야 하므로, txt file로 CRUD 명령어를 작성한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

생성하는 Table은 위에서 설명한 physical schema의 조건을 그대로 따랐으며, Value는 해당 attribute에 알맞은 값을 임의로 설정해 집어넣는다. 자세한 내용은 20171645.txt 파일을 참고한다.

1. **ODBC C language**

**global**

텍스트, 점수판, 닫기이(가) 표시된 사진

자동 생성된 설명

mysql DB와 연결할 정보를 저장하는 문자열로, 각각 host 이름, user 이름, 비밀번호, 연결할 db 이름을 나타낸다.

텍스트이(가) 표시된 사진

자동 생성된 설명

각 query TYPE에 따른 알맞은 query를 실행하는 함수이다.

**main()**

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 mysql\_init() 을 통해 MYSQL 오브젝트를 할당하고 초기화한다.

그뒤, mysql\_real\_connect() 함수를 통해 위에서 설정한 const char\* 변수를 활용해 DB와 실제로 연결한다. mysql\_init 과 mysql\_real\_connect시에 적절히 예외처리를 해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

CRUD query 명령어를 저장한 txt 파일을 불러온다. 프로젝트 내의 “20171645.txt” 파일을 불러와, fgets() 함수를 통해 한줄씩 읽어들여 buffer에 저장하고, 해당 query 명령어를 mysql\_query() 함수를 사용하여 실행한다. 한번의 query가 끝나면 memset() 함수를 통해 buffer를 다시 비워준다. 만약 읽어들인 query 명령어가 DROP TABLE ~ 명령어라면, 해당 query를 실행하기 전 break 문을 통해 while문을 벗어난다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그런 뒤, 사용자 input을 받을 수 있도록 char 변수 input을 선언한 뒤, 형식에 맞게 user interface를 출력한 뒤 \_getch() 함수를 통해 입력을 받는다. 입력받은 값에 따라 알맞은 TYPE의 query를 실행할 수 있도록 if else문을 통해 해당 TYPE의 함수를 호출한다. 만약 입력이 0이라면 while문을 종료한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

만약 user input으로 0을 입력받아 while문을 벗어나면, 먼저 이전의 파일 입출력에서 마지막으로 buffer에 저장되어있던 “DROP TABLE ~” query를 실행한 뒤, fgets() 함수 결과가 null이 될 때까지 한줄씩 파일을 읽고 해당 query를 mysql\_query() 함수로 실행한다.

모든 CRUD query가 끝났다면, flose(fp) 및 mysql\_close(connection)으로 연 파일과 db와 연결된 connection을 종료해준다.

**type\_one()**

텍스트이(가) 표시된 사진

자동 생성된 설명

형식에 맞게 user interface를 출력한 뒤, scanf() 를 통해 찾고자 하는 tracking number를 user input으로 받고, 이를 통해 다음 query 명령어를 실행한다.

select \* from customer c, (select customer\_id from orders odr, (select order\_id from shipment where company\_name = 'USPS' and tracking\_number = X and status = 'Stopped by accident') tracker where odr.order\_id = tracker.order\_id) cid where c.customer\_id = cid.customer\_id;

USPS에 의해 운송되던 tracking number가 X이며 배송의 배송상태가 ‘Stopped by accident’ 인 배송의 고객의 customer\_id를 2중 query 중 내부의 query로부터 가져오고, 외부 query는 해당 customer\_id를 가진 customer의 정보를 select 해오는 query이다. 이를 통해 사고로 파손된 tracking number X를 갖는 USPS의 배송을 시킨 고객의 정보를 얻을 수 있다.

mysql\_store\_result() 함수를 통해 query 결과를 가져오고, mysql\_fetch\_row() 함수를 통해 해당 결과를 각 row 별로 sql\_row에 따로 저장한다. 이때 flag 변수를 통해 query 결과가 하나 이상 있다면 해당 query 결과를 전부 형식에 맞게 출력하고, 하나도 없다면

No Query Result. Please check your tracking number input 를 출력해준다.

해당 query를 수행한 결과는



와 같은 형태로 반환되는데, TYPE 1 query가 요구하는 정보는 customer의 contact information 이므로 고객의 id, 이름, 전화번호만을 출력한다.

현재 구성한 table에서 회사가 USPS이면서 배송상태가 ‘Stopped by accident’ 인 배송의 경우는 tracking number가 7014 하나뿐인데, 이를 입력한 결과는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

이후 TYPE 1의 subtype인 TYPE 1-1 query를 할 지 입력받는다.

동일하게 \_getch() 함수를 사용하여 사용자에게 입력받고, 만약 0을 입력받는다면 SELECT QUERY TYPES 화면으로 돌아가고 1을 입력받으면 SUBTYPE 1-1 query를 수행하도록 한다. TYPE 1-1 query의 경우 아래 query 명령어를 실행한다.

select k.product\_name, m.quantity from product k, (select w.product\_id, w.quantity from ordered\_product w, (select \* from shipment where tracking\_number = X) z where w.order\_id = z.order\_id) m where k.product\_id = m.product\_id

위의 X는 TYPE 1에서 입력받은 X와 동일한 값으로, query 내용은 3중 query중 가장 내부 query를 통해 tracking\_number가 X인 shipment 정보를 가져오고, 그 밖 내부 query를 통해 해당 shipment와 order\_id가 동일한 ordered\_product의 product\_id, quantity를 가져와서, 가장 밖 query를 통해 product\_id가 동일한 product의 이름과, 이전에 가져온 quantity를 출력한다.

그 결과, tracking number가 X인 shipment의 배송 내용물의 이름과 그 개수를 출력한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그런다음, 해당 정보를 바탕으로 새로운 shipment를 replacement items로 준비하기 위해, 아래 query를 수행한다.

select \* from shipment order by tracking\_number desc

해당 query를 실행한 결과, 모든 shipment 내용을 tracking\_number 순으로 내림차순 정렬해 가져오는데,

sql\_result = mysql\_store\_result(connection) 와

sql\_row = mysql\_fetch\_row(sql\_result) 를 통해 해당 query의 첫 줄만 가져오고,

strcpy(temp, sql\_row[0]) 를 통해 가장 큰 tracking number를 구해온다.

현재 구성된 table에서 tracking\_number는 7000부터 시작해서 순차적으로 1씩 올라 7001, 7002, …. 의 형태로 저장되는 형태이므로, 가장 큰 tracking number를 구해 1 더한 값을 할당하면 겹치지 않는 tracking number를 구할 수 있다. atoi() 및 sprintf() 함수를 통해 가장 큰 tracking number + 1의 값을 문자열 형태로 temp에 저장하고, 아래 query를 수행한다.

INSERT INTO Shipment(tracking\_number, company\_name, order\_id, status, origin\_address, current\_address, destination\_address, start\_date, primised\_date, end\_date) SELECT K, s.company\_name, s.order\_id, 'Information', s.origin\_address, s.origin\_address, s.destination\_address, '2022:05:30 12:00:00', '2022:06:02 12:00:00', NULL from shipment s where s.tracking\_number = X

.동일하게 X는 이전의 tracking number이고, K는 바로 전에 구한 biggest tracking number + 1 값이다.

이를 query를 통해, 이전의 tracking number X를 갖는 shipment에서 company\_name, order\_id, origin\_address, destination\_address를 가져오고, K를 새로운 tracking number로, 현재상태를 “information”, 즉 배송정보 접수로, 현재 위치를 origin address와 동일하게, 배송시작 시간, 배송예정 시간을 임의로 설정한 값으로 설정하여, INSERT INTO query 명령어를 통해 Shipment에 새로운 tuple 값을 생성해 집어넣는다.

시간 설정의 경우 time.h library를 활용하여 현재 시간으로 설정해야 할지 고민했으나 프로젝트에 중요한 내용이 아니라고 생각해 임의의 값으로 설정했다.

해당 query 명령어를 통해 tuple을 집어넣는데 성공하면 성공했다고 출력한다.

subquery 실행 결과는 아래와 같이 나온다.

텍스트이(가) 표시된 사진

자동 생성된 설명



새로 생성한 tuple이 정상적으로 들어간 모습을 확인할 수 있다.

**type\_two**

텍스트이(가) 표시된 사진

자동 생성된 설명

아래 query를 실행한다.

select \* from customer q, (select temp.customer\_id from (select customer\_id, sum(total\_price) as s from orders where DATE(order\_date) between '2021-01-01' and '2021-12-31' group by customer\_id) temp order by temp.s desc limit 1) res where q.customer\_id = res.customer\_id

위 query는 orders의 order\_date의 값이 작년, 즉 2021-01-01 부터 2021-12-31인 order들을 customer\_id를 기준으로 group화 하여 각 customer 별로 총액을 구한 뒤 내림차순 정렬한뒤 가장 위의 1개 값을 구하여, 그 customer의 정보를 출력한다.

그 결과, 작년에 주문한 고객들 중 가격 기준으로 가장 많이 구매한 고객 1명을 구한다.

이때, 해당 고객의 customer\_id를 따로 저장한다. (이후 subtype에서 사용)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

이후, subtype 2-1을 TYPE 1-1과 동일한 과정으로 입력받고, 아래 query를 수행한다.

select \* from product pd, (select product\_id from ordered\_product op, (select order\_id, customer\_id from orders where orders.customer\_id = X ) od where op.order\_id = od.order\_id order by op.quantity desc limit 1) pdid where pd.product\_id = pdid.product\_id

이 query는 이전 query에서 구한 customer\_id를 바탕으로, 해당 고객이 구매한 물건을 orderd\_product의 quantity를 기준으로 내림차순 정렬한 뒤 최상위 1개의 값을 구하여 그 고객이 가장 많이 구매한 물건의 product\_id를 구한 뒤, 그 product\_id의 정보를 출력한다.

subquery 실행 결과는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**type\_three()**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

아래 query를 수행한다.

select p.\*, sum(op.quantity), p.price \* sum(op.quantity) as dollar from product p, ordered\_product op, orders o where DATE(o.order\_date) between '2021-01-01' and '2021-12-31' and o.order\_id = op.order\_id and op.product\_id = p.product\_id group by product\_id order by dollar desc

이 query는 order\_date가 작년인 order의 order\_id와 동일한 order\_id를 갖는 ordered\_product의 product\_id를 구한뒤 ordered\_product의 quantity \* price의 sum을 product\_id 를 기준으로 group화 하여 각 물품이 총액 얼마나 팔렸는지에 대한 정보를 구한 뒤 내림차순 정렬하고, 동일한 product\_id를 가진 product의 정보를 출력한다.

그 결과, 작년에 판매된 모든 상품들을 판매 총액 기준 내림차순정렬해 전부 가져와, 출력해준다. 그 결과는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이때 query 결과는 product의 모든 정보를 가져오지만, 모든 내용을 출력하면 출력화면이 너무 지저분해져 product\_id, product\_name, 판매 총액 3가지만 출력하도록 했다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

이후, subtype query를 수행한다. 1을 입력받은 경우, 먼저 사용자 input K를 받은 뒤

select p.\*, sum(op.quantity), p.price \* sum(op.quantity) as dollar from product p, ordered\_product op, orders o where DATE(o.order\_date) between '2021-01-01' and '2021-12-31' and o.order\_id = op.order\_id and op.product\_id = p.product\_id group by product\_id order by dollar desc limit K

query를 통해 TYPE 3의 결과에서 limit K를 통해 상위 K개의 결과만 출력해준다.

2를 입력받은 경우, 먼저

select count(1) from (select p.\*, sum(op.quantity), p.price \* sum(op.quantity) as dollar from product p, ordered\_product op, orders o where DATE(o.order\_date) between '2021-01-01' and '2021-12-31' and o.order\_id = op.order\_id and op.product\_id = p.product\_id group by product\_id order by dollar desc) tb

를 통해, TYPE 3 결과에서 tuple의 총 개수를 구한 뒤, 그 결과를 result에 저장한다.

이후 atoi 함수를 통해 결과를 정수로 바꾼 뒤, 상위 10% product의 개수를 얻기 위해 그 값을 10으로 나눈다. C에서 나눈 값을 int로 type casting할경우 버림이 되는데, 이럴 경우 결과가 0인 경우가 생기므로 if문을 통해 결과가 0일경우 최상위 1개를 출력하도록 설정한다. 이후

select p.\*, sum(op.quantity), p.price \* sum(op.quantity) as dollar from product p, ordered\_product op, orders o where DATE(o.order\_date) between '2021-01-01' and '2021-12-31' and o.order\_id = op.order\_id and op.product\_id = p.product\_id group by product\_id order by dollar desc limit X

를 통해 바로 전에 계산한 상위 10%를 출력한다.

실행 결과는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

**type\_four()**

텍스트이(가) 표시된 사진

자동 생성된 설명

TYPE 4는 query와 2개의 subquery 모두 TYPE3와 거의 모든 면에서 동일하다. 유일한 차이점은, TYPE 4의 query 내용은 아래와 같은데

select p.\*, sum(op.quantity) from product p, ordered\_product op, orders o where DATE(o.order\_date) between '2021-01-01' and '2021-12-31' and o.order\_id = op.order\_id and op.product\_id = p.product\_id group by product\_id order by sum(op.quantity) desc

유일한 차이점은 order by sum(op.quantity)로, 기존의 sum(op.quantity) \* price 로 물건의 판매 총액을 기준으로 정렬한 것을, 물건의 unit sales, 즉 판매 개수 기준으로 정렬한다는 것이 차이점이다. 그 외에는 전부 동일하다.

실행 결과는 각각 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명



**type\_five()**

텍스트이(가) 표시된 사진

자동 생성된 설명

아래 query를 수행한다.

select p.\* from product p, store\_inventory si, store s where s.store\_address = 'California' and s.store\_id = si.store\_id and si.quantity = 0 and si.product\_id = p.product\_id

이 query는 store\_address가 California 이면서 그 store\_id와 동일한 store\_id를 갖는 store\_inventory의 quantity가 0이며 그 store\_inventory의 product\_id와 동일한 product\_id를 갖는 product의 모든 정보를 select 해온다.

그 실행 결과는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**type\_six()**

텍스트이(가) 표시된 사진

자동 생성된 설명

아래 query를 수행한다.

select tracking\_number, company\_name, op.order\_id, p.product\_name, op.quantity from shipment sh, ordered\_product op, product p where sh.primised\_date < sh.end\_date and sh.order\_id = op.order\_id and op.product\_id = p.product\_id group by order\_id

이 query는 shipment의 promised\_date보다 end\_date가 늦은, 즉 약속된 시간보다 배송이 늦게 완료된 shipment의 order\_id, 이를통해 해당 order의 내용물과 주문한 상품 수를 얻어 tracking\_number, company\_name, order\_id, product\_name, quantity를 select 해오는 query 문이다. 그 결과, 약속된 시간보다 배송이 늦게 된 배송과 그에 대응하는 주문, 그 주문의 내용물인 상품의 이름을 가져온다.

실행 결과는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**type\_seven()**

아래 query를 수행한 결과를 출력한다.

select \* from customer c, orders o, ordered\_product op, product p where DATE(o.order\_date) between '2022-05-01' and '2022-06-01' and o.customer\_id = c.customer\_id and o.order\_id = op.order\_id and op.product\_id = p.product\_id order by c.customer\_id

위 query는 order\_date가 저번달인 2022-05-01부터 2022-06-01 사이인 모든 order의 order 정보와 해당 order를 주문한 customer, 해당 order의 내용물 ordered\_product의 각 product의 정보를 전부 select 해오는 query 문이다. 출력시 화면의 가독성을 위해, 필요한 내용인 고객의 id, 고객의 이름, 주문 id, 주문의 총액, 주문의 상품 이름, 주문한 상품 개수를 출력해준다.

해당 query의 결과는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

