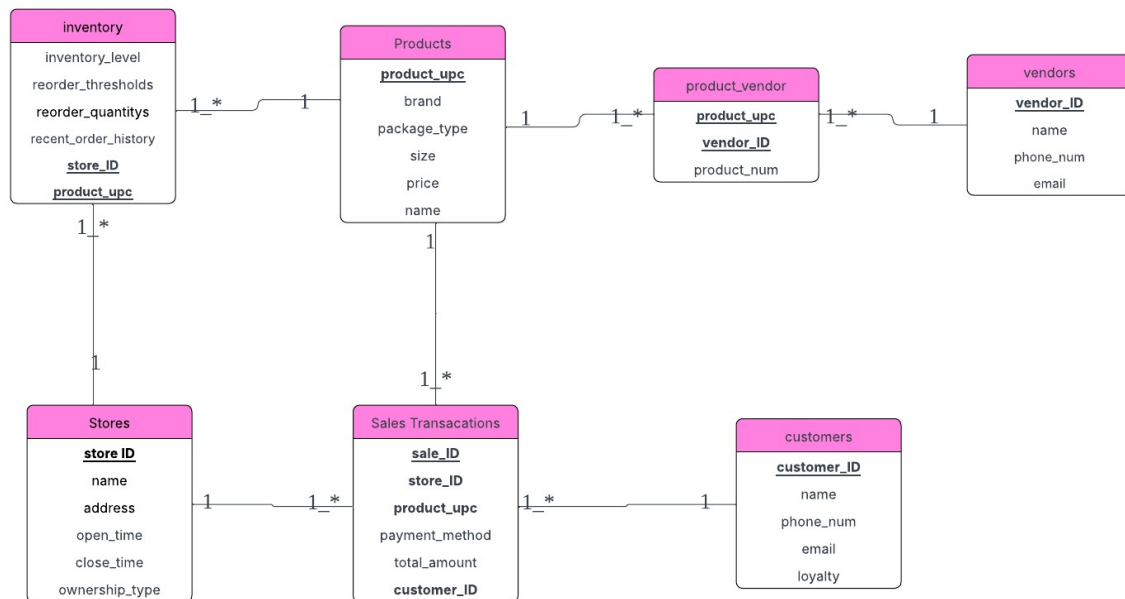


데이터베이스시스템 project2 - Convenience Stores

20221212 이준석

1. Logical Schema Design (BCNF)



(bold체 + 밑줄친 속성 = primary key , bold체만 = foreign key, 나머지 = 일반속성)

1. stores: store_id는 name, address, open_time, close_time, ownership_type을 결정한다.

그리고 다른 일반 속성들간에 함수적 종속성이 없으므로 bcnf를 만족한다.

2. product: product_upc는 name, brand, package_type, size, price를 결정한다.

그리고 다른 일반 속성들간에 함수적 종속성이 없으므로 bcnf를 만족한다.

추가로 erd에는 없었던 name이라는 속성을 이번 과제에서 추가했다.

3. inventory: store_id와 product_upc가 각각 외래키이며, 합쳐서 Primary Key로 설정했다.

그리고 이 primary key는 inventory_level, reorder_thresholds, reorder_quantities, recent_order_history를 결정한다. 그리고 다른 일반 속성들간에 함수적 종속성이 없으므로 bcnf를 만족한다.

4. sales transactions: store_id, product_upc, customer_id 는 각각 외래키이며 따로 sale_id 라는 primary key로 인덱스 하였다. 그리고 다른 일반 속성들간에 함수적 종속성이 없으므로 bcnf를 만족한다. 추가로 erd에는 없었던 buy_date이라는 속성을 추가했다.

5. customer: customer_id는 name, phone_num, email, loyalty를 결정한다.

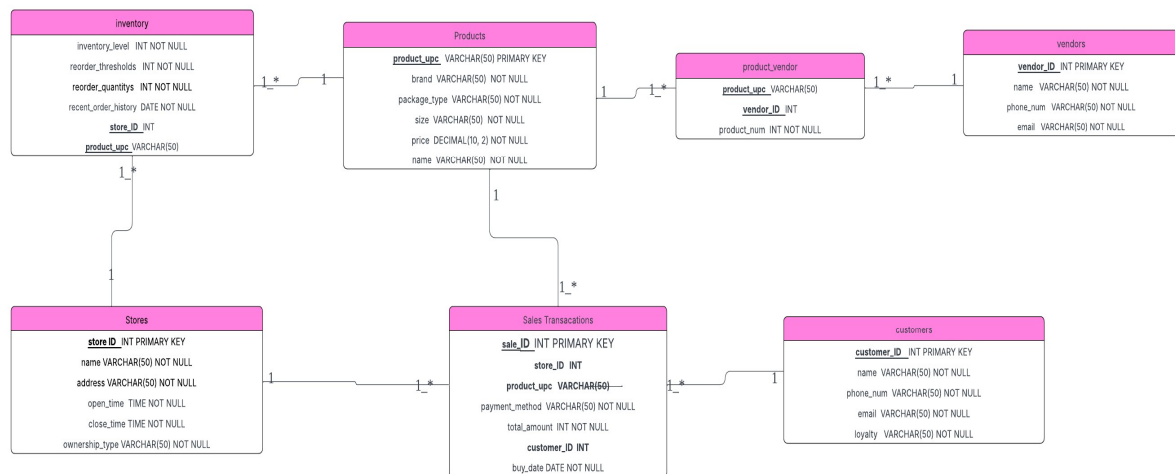
그리고 다른 일반 속성들간에 함수적 종속성이 없으므로 bcnf를 만족한다.

6. vendor: vendor_id는 name, phone_num, email을 결정한다.

그리고 다른 일반 속성들간에 함수적 종속성이 없으므로 bcnf를 만족한다.

7. product_vendor: product_upc, vendor_id는 각각 외래키이며 합쳐서 Primary Key로 설정했다. 이 primary key는 product_num을 결정한다. 그리고 다른 일반 속성들간에 함수적 종속성이 없으므로 bcnf를 만족한다.

2. Physical Schema Design



check관련 내용은 아래 mysql에 적어두었다.

1. stores:

Store_id : 편의점의 id, int 정수 숫자이며 PIRMARY KEY이다.

name: 편의점의 이름 , VARCHAR(50) 문자열이며 NOT NULL 이다.

Address: 편의점의 주소, VARCHAR(50) 문자열이며 NOT NULL 이다.

Open_time: 편의점의 오픈시간 , TIME인 00:00:00으로 표현되며 NOT NULL 이다.

Close time: 편의점의 닫는시간, time 인 00:00:00으로 표현되며 NOT NULL 이다.

Ownership_type: 편의점의 운영주체, VARCHAR(50) 문자열이며 NOT NULL 이다. 프랜차이즈, 직영점, 개인으로 나뉜다.

2. product:

Product_upc: 물건의 id값, VARCHAR(50)인 문자열이며 primary key이다.

Name: 물건의 이름, VARCHAR(50) 문자열이며 NOT NULL 이다.

Brand: 물건의 브랜드, VARCHAR(50) 문자열이며 NOT NULL 이다.

Package_type, 물건의 포장 방식, VARCHAR(50) 문자열이며 NOT NULL 이다. 박스, 병, 팩으로 나뉜다.

Size: 물건의 크기, VARCHAR(50) 문자열이며 NOT NULL 이다. Large, medium, small으로 나뉜다.

Price: 물건의 가격, DECIMAL(10,2) 인 소수 둘째자리까지 나타내는 숫자이며 NOT NULL 이다.

3. inventory:

Store_id: 편의점의 id, int 정수 숫자이며 FOREIGN KEY이다. Stores 테이블에서 참조하고 stores에서 삭제된다면 여기서도 삭제된다.

Product_upc: 물건의 id값, VARCHAR(50)인 문자열이며 foreign key이다. products 테이블에서 참조하고 products에서 삭제된다면 여기서도 삭제된다.

Inventory_level: 현재 재고수량, int 정수 숫자이며 NOT NULL이다. 0이상의값이다.

Reorder_thresholds: 최소 재고 유지 수량, int 정수 숫자이며 NOT NULL이다. 0초과의 값이다.

Reorder_quantity: 주문시 오는 재고 수량, int 정수 숫자이며 NOT NULL이다. 0초과의 값이다.

Recent_order_history: 최근 주문일자, DATE 이며 0000-00-00 식으로 년월일이 표시된다. NOT NULL이다.

Store_id, product_upc를 묶어서 primary key이다.

4. sales transactions:

Sale_id: sale 의 id값, int 정수 숫자이며 PRIMARY KEY이다.

Store_id: 편의점의 id, int 정수 숫자이며 FOREIGN KEY이다. Stores 테이블에서 참조하고 stores에서 삭제된다면 여기서도 삭제된다.

Product_upc: 물건의 id값, VARCHAR(50)인 문자열이며 foreign key이다. products 테이블

에서 참조하고 products에서 삭제된다면 여기서도 삭제된다.

customer_id: 고객의 id, int 정수 숫자이며 FOREIGN KEY이다. customers 테이블에서 참조하고 cusotmers에서 삭제된다면 여기서도 삭제된다.

payment_method: 어떠한 방식으로 구매했는지, VARCHAR(50) 문자열이며 NOT NULL 이다. 현금과 신용카드로 나뉜다.

total_amount: 판매한 물품의 개수, int 정수 숫자이며 NOT NULL이다. 0초과의 값이다.

Buy_date: 물품을 구매한 날짜, date인 0000-00-00형식이며 NOT NULL이다.

Store_id,product_upc,customer_id가 묶여서 primary key의 역할을 할 수 있지만 너무 많아서 따로 sale_id라는 인덱스를 만들었다.

5. customer:

Customer_id: 고객의 id, int 정수 숫자이며 PRIMARY KEY이다.

Name ,고객의 이름, VARCHAR(50) 문자열이며 NOT NULL 이다.

phone_num : 고객의 핸드폰 번호, VARCHAR(50) 문자열이며 NOT NULL 이다.

Email : 고객의 이메일, VARCHAR(50) 문자열이며 NOT NULL 이다.

Loyalty: 고객의 충성도 등급, VARCHAR(50) 문자열이며 NOT NULL 이다. Vip와 일반으로 나뉜다.

6. vendor:

vendor_id 공급업체의 id값, int 정수 숫자이며 PRIMARY KEY이다.

Name: 공급업체의 이름, VARCHAR(50) 문자열이며 NOT NULL 이다.

phone_num: 공급업체의 전화번호, VARCHAR(50) 문자열이며 NOT NULL 이다.

Email: 공급업체의 이메일, VARCHAR(50) 문자열이며 NOT NULL 이다.

7. product_vendor:

product_upc: 물건의 id값, VARCHAR(50) 문자열이며 foreign key이다. products 테이블에서 참조하고 products에서 삭제된다면 여기서도 삭제된다.

vendor_id: 공급업체의 id값 , int 정수 숫자이며 foreign key이다. vendors 테이블에서 참조하고 vendors에서 삭제된다면 여기서도 삭제된다.

product_num: 물건의 공급 개수 , int 정수 숫자이며 NOT NULL이다. 0초과의 값이다.

Product_upc, vendor_id 묶여서 primary key이다.

3. Database Implementation

```
CREATE TABLE stores (  
    store_id INT PRIMARY KEY,  
    name VARCHAR(50) NOT NULL,  
    address VARCHAR(50) NOT NULL,  
    open_time TIME NOT NULL,  
    close_time TIME NOT NULL,  
    ownership_type VARCHAR(50) NOT NULL  
        check (ownership_type in ('Franchise', 'Corporate', 'Independent'))  
);  
  
CREATE TABLE products (  
    product_upc VARCHAR(50) PRIMARY KEY,  
    name VARCHAR(50) NOT NULL,  
    brand VARCHAR(50) NOT NULL,  
    package_type VARCHAR(50) NOT NULL  
        check (package_type in ('Box', 'Bottle', 'Pack')),  
    size VARCHAR(50) NOT NULL  
        check (size in ('Large', 'Medium', 'Small')),  
    price DECIMAL(10, 2) NOT NULL check(price>0)  
);  
  
CREATE TABLE inventory (  
    store_id INT,  
    product_upc VARCHAR(50),  
    inventory_level INT NOT NULL check(inventory_level >=0),  
    reorder_thresholds INT NOT NULL check(reorder_thresholds>0),  
    reorder_quantities INT NOT NULL check(reorder_quantities>0),  
    recent_order_history DATE NOT NULL,  
    PRIMARY KEY (store_id, product_upc),  
    FOREIGN KEY (store_id) REFERENCES Stores(store_id) ON DELETE CASCADE,  
    FOREIGN KEY (product_upc) REFERENCES Products(product_upc) ON DELETE CASCADE  
);  
  
CREATE TABLE customers (  
    customer_id INT PRIMARY KEY,  
    name VARCHAR(50) NOT NULL,  
    phone_num VARCHAR(50) NOT NULL,  
    email VARCHAR(50) NOT NULL,  
    loyalty VARCHAR(50) NOT NULL  
        check(loyalty in ('VIP', 'normal'))  
);  
  
CREATE TABLE sales_Transactions (  
    sale_ID INT PRIMARY KEY,  
    store_id INT,  
    product_upc VARCHAR(50),  
    customer_id INT,  
    payment_method VARCHAR(50) NOT NULL  
        check(payment_method in ('Credit Card', 'Cash')),  
    total_amount INT NOT NULL check(total_amount>0),  
    buy_date DATE NOT NULL,  
    FOREIGN KEY (store_id) REFERENCES Stores(store_id) ON DELETE CASCADE,  
    FOREIGN KEY (product_upc) REFERENCES Products(product_upc) ON DELETE CASCADE,  
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE CASCADE  
);  
  
CREATE TABLE vendors (  
    vendor_id INT PRIMARY KEY,  
    name VARCHAR(50) NOT NULL,  
    phone_num VARCHAR(50) NOT NULL,  
    email VARCHAR(50) NOT NULL  
);
```

```
CREATE TABLE product_Vendor (
    product_upc VARCHAR(50),
    vendor_id INT,
    product_num INT NOT NULL check(product_num>0),
    PRIMARY KEY (product_upc, vendor_id),
    FOREIGN KEY (product_upc) REFERENCES Products(product_upc) ON DELETE CASCADE,
    FOREIGN KEY (vendor_id) REFERENCES Vendors(vendor_id) ON DELETE CASCADE
);
```

위와 같이 table을 create해서 sql파일에 실재하여 table을 생성할 수 있다.

또한 적절한 데이터 샘플을 생성하여 데이터를 insert할 수 있다.

sql파일의 처음 부분에 drop table if exist를 추가하여 기존에 이러한 이름의 테이블이 존재한다면 삭제하는 쿼리를 처음에 넣어준다.

4. Application Development

```
MYSQL* conn;
const char* server = "localhost";
const char* user = "root";
const char* password = "1234"; // 여기에 비밀번호 입력
const char* database = "store"; // 여기에 데이터베이스 이름 입력

// MySQL 초기화
conn = mysql_init(nullptr);
if (conn == nullptr) {
    std::cerr << "mysql_init() failed\n";
    return 1;
}

// MySQL 서버 연결
if (mysql_real_connect(conn, server, user, password, database, 0, nullptr, 0) == nullptr) {
    std::cerr << "mysql_real_connect() failed\n";
    mysql_close(conn);
    return 1;
}
```

Server, user, password, database를 입력하고 mysql_init을 통해 초기화한 뒤 mysql_real_connect를 이용해 연결한다.

```
void query_function(const char* query, MYSQL* conn) {
    MYSQL_RES* res;
    MYSQL_ROW row;

    // 쿼리 실행
    if (mysql_query(conn, query)) {
        std::cerr << "SELECT failed. Error: " << mysql_error(conn) << "\n";
        mysql_close(conn);
    }

    // 결과 저장
    res = mysql_store_result(conn);
    if (res == nullptr) {
        std::cerr << "mysql_store_result() failed. Error: " << mysql_error(conn) << "\n";
        mysql_close(conn);
    }

    // 필드 개수 가져오기
    int num_fields = mysql_num_fields(res);
    MYSQL_FIELD* fields = mysql_fetch_fields(res);

    // 헤더 출력
    for (int i = 0; i < num_fields; i++) {
        std::cout << std::setw(15) << fields[i].name << " ";
    }
    std::cout << "\n";

    // 행 출력
    while ((row = mysql_fetch_row(res))) {
        for (int i = 0; i < num_fields; i++) {
            std::cout << std::setw(15) << (row[i] ? row[i] : "NULL") << " ";
        }
        std::cout << "\n";
    }

    // 리소스 해제
    mysql_free_result(res);
}
```

쿼리와 connection을 입력받는 함수를 만들어서 들어온 쿼리 요청에 대해 쿼리를 출력하는 함수를 만들었다.

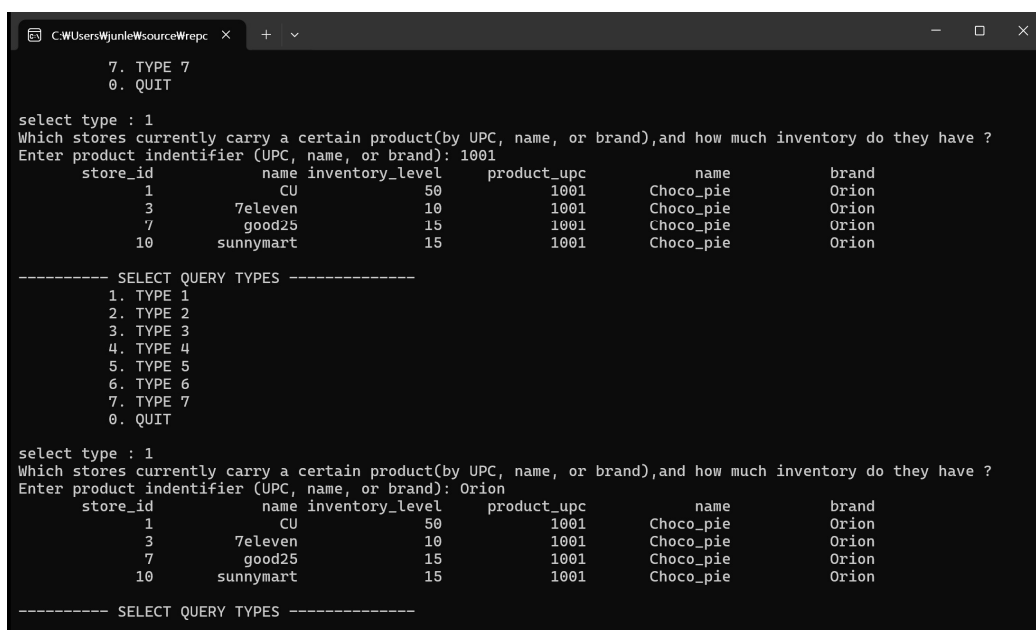
그리고 반복문을 만들고 매번 메뉴화면을 출력하고 아래에 0~7중 값을 입력받고 각각의 값마다 쿼리를 설정해서 쿼리함수를 불러오는 반복문 코드를 만들었다. 예외처리로 0~7 이외의 값이 온다면 다시 반복문의 시작으로 돌아가 메뉴화면을 다시 출력한다.

5. Query Implementation

```
if (N == 1) {
    std::cout << "Which stores currently carry a certain product(by UPC, name, or brand),and how much inventory do they have ?\n";
    std::cout << "Enter product identifier (UPC, name, or brand): ";
    std::string str;
    std::cin >> str;
    std::string s;
    int cnt = 0;
    for (int i = 0; i < str.length(); i++) {
        if (isdigit(str[i])) cnt++;
    }
    if (cnt == str.length()) {
        s = "SELECT s.store_id,s.name,s.inventory_level,p.product_upc,p.name,p.brand\n"
            "FROM products p JOIN (SELECT * FROM inventory NATURAL JOIN stores) s\n"
            "ON p.product_upc=s.product_upc\n"
            "WHERE\n"
            "p.product_upc = " + str;
    }
    else {
        s = "SELECT s.store_id,s.name,s.inventory_level,p.product_upc,p.name,p.brand\n"
            "FROM products p JOIN (SELECT * FROM inventory NATURAL JOIN stores) s\n"
            "ON p.product_upc=s.product_upc\n"
            "WHERE\n"
            "p.name = " + str + " OR\n"
            "p.brand = " + str;
    }
}
```

UPC일때와 name,brand일때를 구분짓는다. 같이 했을 때 문자열 입력시 upc를 확인하는 where 부분에서 오류가 난다.

먼저 재고 테이블과 매장 테이블을 join한 다음 product도 join해주면 product의 어떠한 정보를 얻으면 그 제품이 있는 매장의 id와 재고 수량을 알 수 있다.



```
7. TYPE 7
0. QUIT

select type : 1
Which stores currently carry a certain product(by UPC, name, or brand),and how much inventory do they have ?
Enter product identifier (UPC, name, or brand): 1001
store_id    name    inventory_level    product_upc    name    brand
1           CU      50                1001          Choco_pie    Orion
3           7eleven  10                1001          Choco_pie    Orion
7           good25   15                1001          Choco_pie    Orion
10          sunnymart 15                1001          Choco_pie    Orion

----- SELECT QUERY TYPES -----
1. TYPE 1
2. TYPE 2
3. TYPE 3
4. TYPE 4
5. TYPE 5
6. TYPE 6
7. TYPE 7
0. QUIT

select type : 1
Which stores currently carry a certain product(by UPC, name, or brand),and how much inventory do they have ?
Enter product identifier (UPC, name, or brand): Orion
store_id    name    inventory_level    product_upc    name    brand
1           CU      50                1001          Choco_pie    Orion
3           7eleven  10                1001          Choco_pie    Orion
7           good25   15                1001          Choco_pie    Orion
10          sunnymart 15                1001          Choco_pie    Orion

----- SELECT QUERY TYPES -----
```

```

else if (N == 2) {
    std::cout << "Which products have the highest sales volume in each store over the past month?" << '\n';
    std::string s = " SELECT store_id, p.name, sales FROM products p NATURAL JOIN"
        " (SELECT store_id, product_upc, SUM(total_amount) as sales FROM sales_transactions"
        " WHERE buy_date LIKE W'2025-05%W'"
        " group by store_id, product_upc"
        " HAVING(store_id, sum(total_amount)) IN"
        " (SELECT store_id, MAX(sales) as max_sales FROM("
        " SELECT store_id, product_upc, SUM(total_amount) as sales FROM sales_transactions"
        " WHERE buy_date LIKE W'2025-05%W'"
        " group by store_id, product_upc) s group by store_id)) s";
    const char* query = s.c_str();
    query_function(query, conn);
}

```

매매 테이블에서 store_id, product_upc 그룹화해서 저번달의 판매량의 합을 구하고 각 매장마다 가장 많이 팔린 양을 max로 구하고 having을 이용해 매장마다 가장 많이 팔린 물건들을 구한다.

```

select type : 2
Which products have the highest sales volume in each store over the past month?

```

store_id	name	sales
1	Shin_Ramyun	4
2	kimbab	3
3	Choco_pie	4
4	suncream	4
5	gum	1
6	suncream	9
7	coffee	3
8	Pokari	3
9	suncream	4
10	gum	4

```

else if (N == 3) {
    std::cout << "Which store has generated the highest overall revenue this quarter ?" << '\n';
    std::string s = "SELECT store_id, name, revenue FROM stores s NATURAL JOIN"
        " (SELECT store_id, SUM(total_amount * price) as revenue FROM"
        " (SELECT * FROM sales_transactions t NATURAL JOIN products p"
        " WHERE t.buy_date LIKE W'2025-04%W' OR t.buy_date LIKE W'2025-05%W' OR t.buy_date LIKE W'2025-06%W') sub"
        " GROUP BY store_id ORDER BY revenue DESC LIMIT 1) r";
    const char* query = s.c_str();
    query_function(query, conn);
}

```

올해 쿼터 즉 4월 5월 6월에 팔린 양을 store_id로 그룹화하고 sum한다. 그 후 판매량을 기준으로 내림차순으로 정렬해주고 limit 1을 이용해 상위 하나의 store를 출력한다.

```

select type : 3
Which store has generated the highest overall revenue this quarter ?

```

store_id	name	revenue
6	hanaro	1388.77


```

else if (N == 4) {
    std::cout << "Which vendor supplies the most products across the chain, and how many total units have been sold ? " << '\n';
    std::string s = " SELECT vendor_id, name, supply, sales FROM vendors v NATURAL JOIN"
                    " (SELECT vendor_id, sum(supply) as supply, sum(sales) as sales FROM"
                    " (SELECT vendor_id, product_upc, SUM(product_num) AS supply FROM product_vendor"
                    " GROUP BY vendor_id, product_upc) supply NATURAL JOIN"
                    " (SELECT product_upc, SUM(total_amount) AS sales FROM sales_transactions"
                    " GROUP BY product_upc) sale"
                    " GROUP BY vendor_id"
                    " ORDER BY supply DESC LIMIT 1) sub";
    const char* query = s.c_str();
    query_function(query, conn);
}

```

Product_vendor 테이블에서 vendor_id로 그룹화해서 총 공급량을 구하고 limit 1을 이용해 가장 공급량이 큰 vendor를 출력한다. 매매 테이블에서 판매량을 구하고 product와 join해서 x가격의 합을 구하여 가장 공급량이 큰 vendor의 판매량을 구한다.

```

select type : 4
Which vendor supplies the most products across the chain, and how many total units have been sold
  vendor_id      name      supply      sales
    5           kakao        450         17

```

```

else if (N == 5) {
    std::cout << "Which products in each store are below the reorder threshold and need restocking ? " << '\n';
    std::string s = "SELECT store_id, product_upc, inventory_level, reorder_thresholds"
                    " FROM inventory WHERE inventory_level < reorder_thresholds";
    const char* query = s.c_str();
    query_function(query, conn);
}

```

재고 테이블에서 inventory_level < reorder_threseholds를 이용하여 재고가 부족한 store와 물품의 정보를 출력한다.

```

select type : 5
Which products in each store are below the reorder threshold and need restocking ?
  store_id      product_upc  inventory_level  reorder_thresholds
    3           1001           10                20
    3           1002           20                30
    6           1002           20                25
    6           1005           10                15
    8           1005           10                20

```

```

else if (N == 6) {
    std::cout << "List the top 3 items that loyalty program customers typically purchase with coffee." << '\n';
    std::string s = " SELECT product_upc, name, sales FROM products p NATURAL JOIN"
        " (SELECT product_upc, SUM(total_amount) as sales FROM sales_transactions"
        " WHERE(store_id, payment_method, buy_date, customer_id) IN"
        " (SELECT vip.store_id, vip.payment_method, vip.buy_date, vip.customer_id FROM products p JOIN"
        " (SELECT * FROM sales_transactions NATURAL JOIN customers"
        " WHERE loyalty = W\"VIPW\") vip ON p.product_upc = vip.product_upc"
        " WHERE p.name = W\"coffeeW\")"
        " GROUP BY product_upc) sub"
        " HAVING name <> W\"coffeeW\""
        " ORDER BY sales DESC LIMIT 3";
    const char* query = s.c_str();
    query_function(query, conn);
}

```

Loyalty 등급이 VIP인 사람들의 정보를 알아낸뒤 매매 테이블과 product테이블을 join해 이 정보를 이용한 사람이 coffee를 결제했을때의 store_id,payment_method,buy_date,customer_id를 구해서 이와 같은 거래를 했을때의 총 합을 sum을 이용하여 구한다. 이때 물품이름이 coffee일 때를 빼야한다. 판매량을 기준으로 내림차순으로 정렬하고 LIMIT 3을 이용해 탑3 제품을 출력한다.

```

select type : 6
List the top 3 items that loyalty program customers typically purchase with coffee.

```

product_upc	name	sales
1009	gum	3
1004	kimbab	2
1001	Choco_pie	1

```

else if (N == 7) {
    std::cout << "Among franchise-owned stores, which one offers the widest variety of products, and how does that compare to corporate-owned stores ?"
    std::string s = "SELECT ownership_type, ROUND(AVG(variety),2) AS variety"
        " FROM stores NATURAL JOIN"
        " (SELECT store_id, COUNT(*) AS variety FROM inventory GROUP BY store_id) v"
        " GROUP BY ownership_type"
        " ORDER BY variety DESC";
    const char* query = s.c_str();
    query_function(query, conn);
}

```

재고 테이블에서 store_id마다의 count(*)을 구해서 다양성을 구한다. Store와 join을 한 후 ownership_type마다의 평균 다양성을 구한다.

```

select type : 7
Among franchise-owned stores, which one offers the widest variety of products, and how does that compare to corporate-owned stores ?

```

ownership_type	variety
Corporate	3.33
Franchise	2.80
Independent	1.00