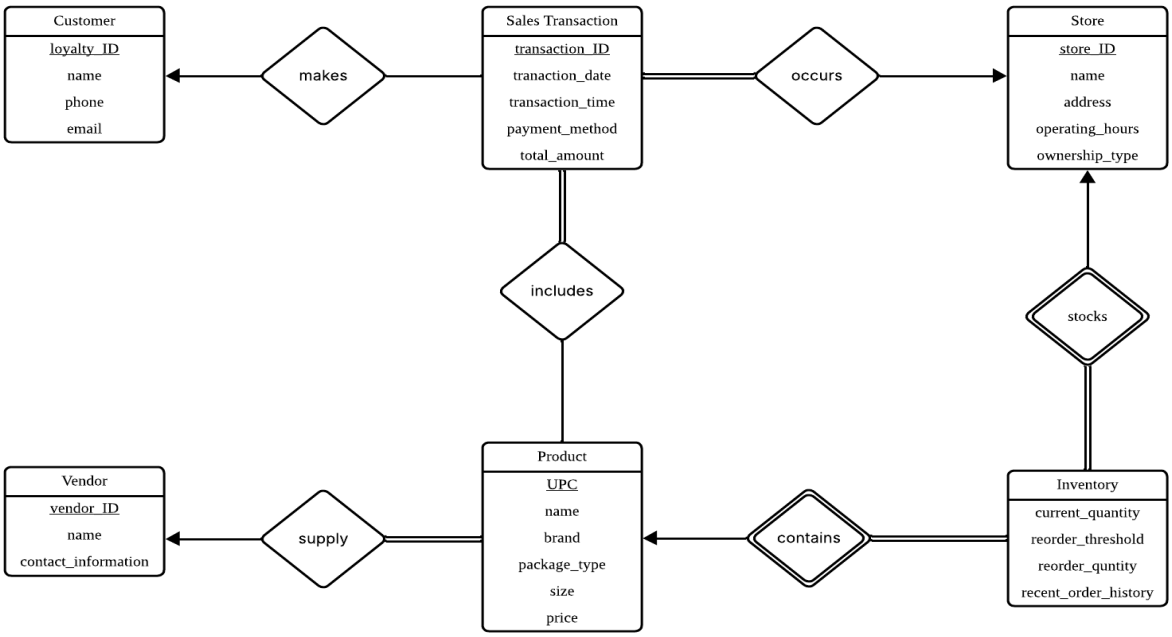
CSE4110 - Database System

DB Project 2 Report

20190808 방지혁

1. Reduction to Relational Schema

아래는 project1에서의 ER diagram



1-1) Strong Entity 변경

Strong Entity Set 부터 변경한다. – 밑줄 쳐진 것이 primary key

Customer (loyalty_ID, name, phone, email)

Sales_Transaction (transaction_ID, transaction_date, transaction_time, payment_method, total_amount)

Store (store_ID, name, address, operating_hours, ownership_type)

Product (UPC, name, brand, package_type, size, price)

Vendor (vendor_ID, name, contact_information)

→ **Vendor** (vendor_ID, name, phone, email)로 변경: contact_information 은 애매하기 때문에 그렇게 결정했다.

1-2) 1 : M Relationship 변경 – makes, supply, occurs (many side 에 one side 의 primary key 삽입)

Supply 의 경우) product table 에 vendor_ID 추가(foreign key)

Makes 의 경우) sales_transaction table 에 loyalty_ID 추가(foreign key)

Occurs 의 경우) sales_transaction table 에 store_ID 추가(foreign key)

Result)

Sales_Transaction (transaction_ID, transaction_date, transaction_time, payment_method, total_amount, loyalty_ID, store_ID)

설명 - primary key: transaction_ID, foreign key: loyalty_ID, store_ID

Product (UPC, name, brand, package_type, size, price, vendor_ID)

설명 - primary key: UPC, foreign key: vendor_ID

1-3) Weaky Entity 변경

Weaky entity 인 inventory 를 식별하는 개체인 product 와 store 의 primary key 를 inventory 에 primary key 이자 foreign key 로서 삽입

Inventory (store_ID, UPC, current_quantity, reorder_threshold, reorder_quantity, recent_order_history)

설명 - primary key: store_ID, UPC, foreign key: store_ID, UPC

1-4) M : N Relationship 변경 - includes

Transaction_product라는 새로운 table 만들기

Transaction_product (transaction_ID, UPC, quantity)

+) 쿼리 예측 결과 project 1의 진행 상태로는 count()를 해도 거래횟수만 세고, 실제 판매된 quantity는 모르기에 quantity attribute를 추가해준다.

설명 - primary key: transaction_ID, UPC, foreign key: transaction_ID, UPC

<Conclusion>

Customer (loyalty_ID, name, phone, email)

Sales_Transaction (transaction_ID, transaction_date, transaction_time, payment_method, total_amount, loyalty_ID, store_ID)

Store (store_ID, name, address, operating_hours, ownership_type)

Product (UPC, name, brand, package_type, size, price, vendor_ID)

Vendor (vendor_ID, name, phone, email)

Inventory (store_ID, UPC, current_quantity, reorder_threshold, reorder_quantity, recent_order_history)

Transaction_product (transaction_ID, UPC, quantity)

2. BCNF Normalization

2-1) Customer (loyalty_ID, name, phone, email), Primary Key: loyalty_ID

함수 종속성: loyalty_ID → name, phone, email

loyalty_ID는 primary key이므로 superkey인데 다른 비자명한 함수 종속성은 존재하지 않기에 BCNF를 만족한다.

2-2) Store (store_ID, name, address, operating_hours, ownership_type), Primary Key: store_ID

함수 종속성: store_ID → name, address, operating_hours, ownership_type

store_ID는 primary key이므로 superkey인데 다른 비자명한 함수 종속성은 존재하지 않기에 BCNF를 만족한다.

2-3) Vendor (vendor_ID, name, phone, email), Primary Key: vendor_ID

함수 종속성: vendor_ID → name, phone, email

vendor_ID는 primary key이므로 superkey인데 다른 비자명한 함수 종속성은 존재하지 않기에 BCNF를 만족한다.

2-4) Product (UPC, name, brand, package_type, size, price, vendor_ID)

함수 종속성: UPC → name, brand, package_type, size, price

같은 product를 여러 vendor에서 공급할 수 있기에 1 : M 관계여서 Vendor_ID는 빠진다.

UPC는 primary key이므로 superkey인데 다른 비자명한 함수 종속성은 존재하지 않기에 BCNF를 만족한다.

2-5) Sales_Transaction (transaction_ID, transaction_date, transaction_time, payment_method, ~~total_amount~~, loyalty_ID, store_ID), Primary Key: transaction_ID

함수 종속성: transaction_ID → transaction_date, transaction_time, payment_method, total_amount, loyalty_ID, store_ID

transaction_ID는 primary key이므로 superkey인데 다른 비자명한 함수 종속성은 존재하지 않기에 BCNF를 만족한다.

추가 변경한 부분!! Total_amount 속성은 실제 구매 금액과 안 맞을 수 있다. 그러나 해당 속성에 대한 Check 제약 조건은 단일 테이블 내에서만 작동하기에 검증하기 힘들기 때문에 total_amount 속성을 삭제했다.

2-6) Inventory (store_ID, UPC, current_quantity, reorder_threshold, reorder_quantity, recent_order_history)

Primary key: store_ID, UPC

함수 종속성: (store_ID, UPC) → current_quantity, reorder_threshold, reorder_quantity, recent_order_history

점검할 부분: UPC로부터 함수종속성은 성립하는지 확인해본다. 그러나 같은 제품일지라도 매장의 inventory별로 다른 재주문 threshold, quantity를 가질 것이기에, 앞서 언급한 부분 외에는 다른 비자명한 함수 종속성이 성립하지 않는다.

2-7) Transaction_product (transaction_ID, UPC, quantity), primary key: transaction_ID, UPC

함수 종속성: (transaction_ID, UPC) → quantity

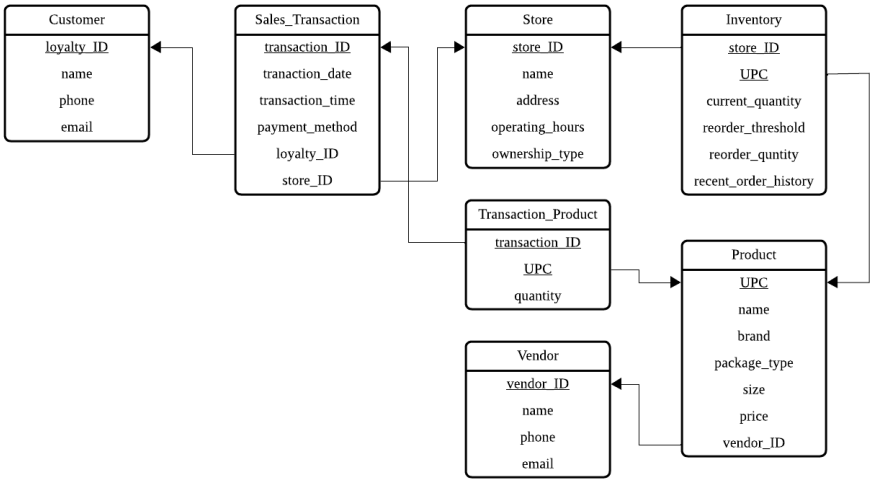
점검할 부분: transaction_ID → quantity

➔ 특정 거래에서 각 제품을 다른 수량으로 구매 가능하기 때문에 성립하지 않는다.

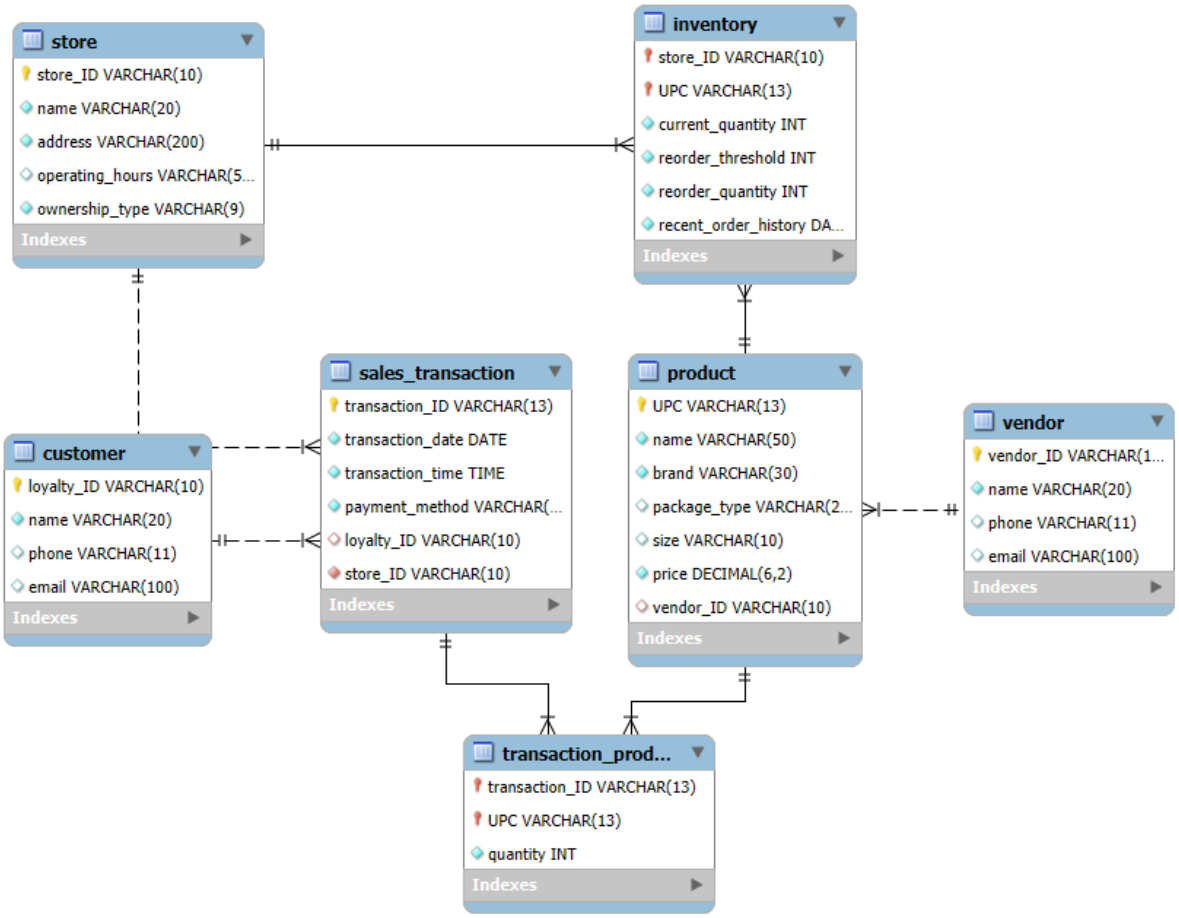
점검할 부분: UPC → quantity

➔ 앞선 부분과 마찬가지로 비슷한 이유인데, 각각의 다른 거래에서 같은 수량으로 항상 판매되는 것이 아니기에 성립하지 않는다.

Final Relational schema



3. Physical Schema Design



설계자 본인은 MySql Workbench에서 Manual EER Diagram을 이용해 수동으로 테이블을 그리는 방법 대신 우선 sql로 각 table에 대한 정의를 하고 이를 reverse engineer를 이용해 diagram을 자동 생성해 이를 PNG 파일로 export 했다.

Relational schema에서 각 table 구현에 대한 설명: schema.sql

CREATE TABLE customer (loyalty_ID VARCHAR(10) NOT NULL, name VARCHAR(20) NOT NULL, phone VARCHAR(11), email VARCHAR(100), PRIMARY KEY (loyalty_ID));	loyalty_ID: 로열티 프로그램에 등록된 customer의 primary key로 NOT NULL이어야 하며 C00x 형식이다. name: 고객 이름으로 20자로 제한했으며 NOT NULL 조건 때문에 customer가 등록되기 위해서 이름이 반드시 입력되어야 한다. phone: 11자인 한국 휴대폰 번호 형식에 맞추었다. email: 100자로 설정하여 일반적인 사용자의 이메일 주소가 사이즈를 초과하여 저장되지 않도록 하였다. phone과 email 정보에 대해서는 NULL을 허용했다.

CREATE TABLE vendor (vendor_ID VARCHAR(10) NOT NULL, name VARCHAR(20) NOT NULL, phone VARCHAR(11), email VARCHAR(100), PRIMARY KEY (vendor_ID));	vendor_ID: 공급자에 대한 primary key로 V00x 형식이다. name은 공급업체명으로 20자로 설정했다. Phone과 email 같은 연락처 정보는 customer와 같은 형식으로 통일성을 유지했다.

CREATE TABLE store (store_ID VARCHAR(10) NOT NULL, name VARCHAR(20) NOT NULL, address VARCHAR(200) NOT NULL, operating_hours VARCHAR(50), ownership_type VARCHAR(9) CHECK (ownership_type in ('franchise', 'corporate')) NOT NULL, PRIMARY KEY (store_ID));	store_ID: 매장에 대한 primary key로 S00x 형식이다. name: 매장의 이름으로 NOT NULL 조건을 추가하였다. address: 200자로 설정하여 상세 주소가 초과되지 않고 저장되도록 하였다. operating_hours: 50자로 운영시간이 “24hours”, "06:00-24:00"와 같이 다양한 형태로 저장될 수 있도록 했다. ownership_type: CHECK로 franchise 혹은 corporate로만 설정되도록 integrity constraint를 설정하였다.

CREATE TABLE product (
  UPC VARCHAR(13) NOT NULL,
  name VARCHAR(50) NOT NULL,
  brand VARCHAR(30) NOT NULL,
  package_type VARCHAR(20),
  size VARCHAR(10),
  price numeric(6,2) NOT NULL CHECK (price > 0),
  vendor_ID VARCHAR(10),
 	
  PRIMARY KEY (UPC),
 	
  FOREIGN KEY (vendor_ID) REFERENCES vendor(vendor_ID)
  on delete set null
  on update cascade
);
	UPC: 국제 표준 바코드인 13자리를 지원하도록 크기를 설정하였다.
	name: 50자로 긴 제품명을 저장할 수 있도록 하였다.
	brand: 30자로 긴 브랜드명을 저장할 수 있도록 하였다.
	price: NUMERIC(6,2)로 설정하여 만 원대 이상의 제품이 없으며 0원 초과이어야 한다고 가정하고 편의점 가격대를 설정하였다.
	
	또한, 외래키인 vendor_ID에 대하여 ON DELETE SET NULL울 설정하여 공급자가 삭제되더라도 제품 정보는 저장되도록 하였으며, ON UPDATE CASCADE을 설정하여 공급자의 ID가 변경될 시 이 또한 자동으로 반영되도록 하였다.

CREATE TABLE sales_transaction (
  transaction_ID VARCHAR(13) NOT NULL,
  transaction_date DATE NOT NULL,
  transaction_time TIME NOT NULL,
  payment_method VARCHAR(4) CHECK (payment_method in ('cash', 'card')) NOT NULL,
  total_amount numeric(8,2) NOT NULL CHECK (total_amount > 0),
  loyalty_ID VARCHAR(10),
  store_ID VARCHAR(10) NOT NULL,
 	
  PRIMARY KEY (transaction_ID),
 	
  FOREIGN KEY (loyalty_ID) REFERENCES customer(loyalty_ID)
  ON DELETE SET NULL
  ON UPDATE CASCADE,
  FOREIGN KEY (store_ID) REFERENCES store(store_ID)
  ON DELETE RESTRICT
  ON UPDATE CASCADE
);
	transaction_ID: 각 거래를 구분하는 primary key로 설정하였다.
	transaction_date/time: 각 거래의 date, time을 기록한다.
	payment_method: CHECK조건을 설정해서 실제 결제 상황을 고려하여 cash 혹은 현금과 같은 결제 수단만 허용하였다.
	total_amount: NUMERIC(8,2)로 편의점에서 보통 한 번의 거래당 액수를 예상하여 설정하였다.
	loyalty_ID: 외래키로 ON DELETE SET NULL, ON UPDATE CASCADE로 설정하여 고객이 탈퇴하여도 장부 상 거래 기록은 중요하기에 보존되도록 설정하였다. 또한, 로열티 등록 고객만 거래하는 것은 아니기에 NULL을 허용했다.
	store_ID: 외래키로 매장이 삭제되어도 거래 기록 보호를 위해 삭제를 금지하였다. 이미 store_ID에 NOT NULL 조건이 걸려있기에 ON DELETE RESTRICT로 설정하였다.

CREATE TABLE inventory (store_ID VARCHAR(10) NOT NULL, UPC VARCHAR(13) NOT NULL, current_quantity INT CHECK (current_quantity >= 0) NOT NULL, reorder_threshold INT CHECK (reorder_threshold >= 0) NOT NULL, reorder_quantity INT CHECK (reorder_quantity > 0) NOT NULL, recent_order_history DATE NOT NULL, PRIMARY KEY (store_ID, UPC), FOREIGN KEY (store_ID) REFERENCES store(store_ID) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (UPC) REFERENCES product(UPC) ON DELETE CASCADE ON UPDATE CASCADE);	Composite primary key이자 foreign key인 (store_ID, UPC)로 구성했다. current_quantity: 현재 재고량이기에 INT로 설정하였고 음수 불가능 제약 조건을 걸었다. reorder_threshold: 재주문 기준점으로 INT로 설정하였고 마찬가지로 같은 조건을 걸었다. reorder_quantity: 재주문 수량이기에 INT로 설정하였고 0 및 음수 불가능 제약 조건을 걸었다. recent_order_history: 최근 주문 일자를 추적해야 하기에 DATE 자료형으로 설정했다.

CREATE TABLE transaction_product (transaction_ID VARCHAR(13) NOT NULL, UPC VARCHAR(13) NOT NULL, PRIMARY KEY (transaction_ID, UPC), FOREIGN KEY (transaction_ID) REFERENCES sales_transaction(transaction_ID) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (UPC) REFERENCES product(UPC) ON DELETE RESTRICT ON UPDATE CASCADE);	(transaction_ID, UPC): Composite primary key이자 foreign key로 구성했다. transaction_ID에 delete 및 update CASCADE 조건을 걸어 거래가 삭제될 경우 같이 삭제되도록 했다. UPC에는 DELETE RESTRICT로 제품이 삭제되어도 거래 기록은 장부상 중요하니 보호되도록 했다.

4. Database Implementation

1) Which stores currently carry a certain product (by UPC, name, or brand), and how much inventory do they have? 특정 제품 (UPC로 확인)을 현재 취급하는 매장과 그 재고량은?

UPC를 기준으로 판단하는 것으로 설정했다. C++ 코드에서는 미리 UPC 배열을 선언하고 새로운 인터페이스를 만들어서 입력 값이 해당 배열에 없으면 다시 입력하거나 exit을 눌러 해당 쿼리에서 나갈 수 있도록 했습니다.

SELECT
  s.store_ID,
  s.name AS store_name,
  p.UPC,
  p.name AS product_name,
  p.brand,
  i.current_quantity
FROM inventory AS i
JOIN store AS s ON i.store_ID = s.store_ID
JOIN product AS p ON i.UPC = p.UPC
WHERE i.UPC = '1000000000001'
  AND i.current_quantity > 0
ORDER BY i.current_quantity DESC;
	inventory 테이블을 store 테이블과 store_ID로 조인하고 product 테이블을 UPC로 조인했습니다.
	제약조건에 대해 설명하자면 i.UPC = '1000000000001': 와 같이 특정 상품만 선택할 수 있도록 했고, i.current_quantity > 0를 통해 재고가 있는 매장만 뽑도록 했습니다.
	또한, 재고량이 많은 순서대로 내림차순 정렬을 했습니다.
	가장 중요한 UPC 조건 부분은 입력한 값마다 다르게 들어가야 하는데 이는 sprintf로 해결했습니다.

```
Select: 1
------- TYPE 1 -------
** Which stores currently carry a certain product (by UPC, name, or brand), and how much inventory do they have? **
valid UPC range: 1000000000001 ~ 1000000000060
select UPC or select exit to exit this query
select: 2342134
not in valid range

valid UPC range: 1000000000001 ~ 1000000000060
select UPC or select exit to exit this query
select: 1000000000001
store_ID                store_name              UPC                     product_name            brand                   current_quantity
S004                    GS25 Sangsu             1000000000001           Americano Coffee        Georgia                 60
S006                    GS25 Sogang Univ        1000000000001           Americano Coffee        Georgia                 55
S003                    SevenEleven Hapjeong    1000000000001           Americano Coffee        Georgia                 50
S001                    GS25 Hongik Univ        1000000000001           Americano Coffee        Georgia                 45
S005                    CU Mangwon              1000000000001           Americano Coffee        Georgia                 40
S002                    CU Sinchon              1000000000001           Americano Coffee        Georgia                 35
```

2) Which products have the highest sales volume in each store over the past month?

지난 달 각 매장에서 판매량이 가장 높은 제품은? "5월 1일-5월 31일로 가정했다.

```
SELECT
    s.store_ID,
    s.name AS store_name,
    p.name AS product_name,
    p.brand,
    SUM(tp.quantity) AS total_sales
FROM sales_transaction AS st
JOIN transaction_product AS tp ON st.transaction_ID = tp.transaction_ID
JOIN product AS p ON tp.UPC = p.UPC
JOIN store AS s ON st.store_ID = s.store_ID
WHERE st.transaction_date BETWEEN '2025-05-01' AND '2025-05-31'
GROUP BY s.store_ID, p.UPC
HAVING SUM(tp.quantity) = (
    SELECT MAX(total_qty)
    FROM (
        SELECT SUM(tp2.quantity) AS total_qty
        FROM sales_transaction AS st2
        JOIN transaction_product AS tp2 ON st2.transaction_ID = tp2.transaction_ID
        WHERE st2.store_ID = s.store_ID
          AND st2.transaction_date BETWEEN '2025-05-01' AND '2025-05-31'
        GROUP BY tp2.UPC
    ) AS store_totals
);
```

Sales_transaction, transaction_product, product, store 4개의 테이블을 연결했다. 채점이 6월에 진행되기에 기간을 제약조건에서 볼 수 있다시피
WHERE st.transaction_date BETWEEN '2025-05-01' AND '2025-05-31' 이렇게 설정하였다.
sales_transaction 테이블과 transaction_product 테이블을 transaction_ID로 JOIN하고, product와 store에 대한 정보도 필요하기에 각각 UPC와 store_ID를 이용하여 JOIN 했다. Store와 product별로 그룹화하고 Sum 함수를 사용해서 각 store, product별로 총 매출을 계산했다. 그러나 그 중 제일 높은 판매량을 알기 위해 having 절을 사용했다. 서브 쿼리에서는 한 매장의 모든 판매량을 계산해서 MAX함수를 사용하여 최고치를 얻어내고 이 값과 일치하는 product만 도출되도록 했다.

```
Select: 2
------- TYPE 2 -------
**  Which products have the highest sales volume in each store over the past month? **
```

store_ID	store_name	product_name	total_sales
S001	GS25 Hongik Univ	Chocolate Cookies	2
S001	GS25 Hongik Univ	Energy Drink	2
S002	CU Sinchon	Cafe Latte	2
S002	CU Sinchon	Chocolate Cookies	2
S002	CU Sinchon	Coca Cola	2
S003	SevenEleven Hapjeong	Honey Butter Chips	5
S004	GS25 Sangsu	Sprite	2
S005	CU Mangwon	Americano Coffee	1
S005	CU Mangwon	Iced Coffee	1
S005	CU Mangwon	Corn Chips	1
S005	CU Mangwon	Croissant	1
S005	CU Mangwon	Donut Chocolate	1
S005	CU Mangwon	Club Sandwich	1
S005	CU Mangwon	Green Tea	1
S005	CU Mangwon	Shin Ramyun	1
S005	CU Mangwon	Buldak Ramyun	1
S005	CU Mangwon	Ice Cream Bar	1
S006	GS25 Sogang Univ	Donut Chocolate	3

3) Which store has generated the highest overall revenue this quarter?

이번 분기에 가장 높은 총 매출을 올린 매장은?" 6월에 수행되므로 2분기(4 ~ 6월)로 가정했습니다.

SELECT s.store_ID, s.name AS store_name, SUM(tp.quantity * p.price) AS total_revenue FROM sales_transaction AS st JOIN transaction_product AS tp ON st.transaction_ID = tp.transaction_ID JOIN product p ON tp.UPC = p.UPC JOIN store s ON st.store_ID = s.store_ID WHERE st.transaction_date >= '2025-04-01' AND st.transaction_date <= '2025-06-30' GROUP BY s.store_ID ORDER BY total_revenue DESC LIMIT 1;	해당 쿼리는 sales_transaction, transaction_product, 채점이 6월에 진행되기 때문에 2분기는 4월부터 6월이라고 가정해서 기간을 설정했다. 우선 sales_transaction 테이블과 transaction_product를 거래 primary key인 transcation_ID로 JOIN하고 product와 store에 대한 정보도 필요하기에 UPC와 store_ID를 기준으로 JOIN 했다. SUM(tp.quantity * p.price) 집계 함수를 사용해서 각 매장별로 총 매출을 계산하고, GROUP BY로 매장별로 그룹화했다. 가장 높은 매출을 가진 매장을 첫번째 row에 뜨게 하기 위해서 ORDER BY total_revenue DESC로 내림차순으로 정렬하였다.

```
Select: 3
------- TYPE 3 -------
**  Which store has generated the highest overall revenue this quarter? **
store_ID                store_name              total_revenue
S001                     GS25 Hongik Univ          150000.00
```

경제가 대공황이라고 가정한다.

4) "Which vendor supplies the most products across the chain, and how many total units have been sold?"

"가장 다양한 제품을 공급하는 vendor와 총 판매 유닛 수는?"

```
SELECT
    v.vendor_ID,
    v.name AS vendor_name,
    COUNT(DISTINCT p.UPC) AS product_diversity_count,
    SUM(tp.quantity) AS total_sold_quantity
FROM vendor AS v
JOIN product AS p ON v.vendor_ID = p.vendor_ID
LEFT JOIN transaction_product AS tp ON p.UPC = tp.UPC
GROUP BY v.vendor_ID
ORDER BY product_diversity_count DESC
LIMIT 1;
```

해당 쿼리를 통해 다양한 제품을 공급하는 vendor를 찾고, 그 vendor의 실제 판매 유닛 수를 분석하고자 한다. Vendor 테이블을 vendor_ID로 product 테이블과 join 해서 실제 공급하는 제품이 있는 vendor에 대한 정보를 얻는다. 이후 transaction_product와는 left join을 했는데 아직 판매되지 않는 제품이 다양성을 계산할 때 누락되는 것을 방지하기 위해서이다.

COUNT(DISTINCT p.UPC)로 vendor가 공흡하는 제풍의 다양성을 계산했고, 총 판매 유닛 수는 SUM(tp.quantity)로 총 수량을 집계했다. GROUP BY v.vendor_ID를 통해 vendor별로 이를 산출했고, 앞서 구한 제품 다양성 기준으로 내림차순 정렬을 하여 LIMIT 1을 한다면 가장 다양한 제품을 보유한 vender를 1개 추출해낼 수 있다.

보다시피 아래는 질의의 결과인데 LIMIT 1을 해제했을 때의 총 판매 수 및 다양성과 비교해본다면 다양한 종류의 제품을 판매하더라도 총 판매 유닛 수는 꼭 비례하여 많지 않다는 것을 확인 할 수 있다.

```
Select: 4
------- TYPE 4 -------
** Which vendor supplies the most products across the chain, and how many total units have been sold **
vendor_ID               vendor_name             product_diversity_count  total_sold_quantity
V012                    Procter Gamble          10                       13
```

vendor_ID	vendor_name	product_diversity_count	total_sold_quantity
V012	Procter Gamble	10	13
V013	CU Central Kitchen	7	43
V005	Lotte Confectionery	6	5
V014	GS25 Food	6	48
V001	Coca Cola Korea	5	99
V008	Nongshim	5	15
V009	Ottogi	4	12
V004	Orion Corporation	3	30
V015	Seven Eleven Korea	3	4
V002	Lotte Chilsung	2	NULL
V003	Starbucks Korea	2	34
V006	Haitai Confectionery	2	28
V007	Crown Confectionery	2	NULL
V011	Yuhan Kimberly	2	NULL
V010	Samyang Foods	1	3

5) Which products in each store are below the reorder threshold and need restocking?

각 매장에서 재주문 임계값 이하로 떨어져 재주문이 필요한 제품은?

SELECT s.store_ID, s.name AS store_name, p.name AS product_name, i.current_quantity, i.reorder_threshold, i.reorder_quantity FROM inventory AS i JOIN store AS s ON i.store_ID = s.store_ID JOIN product AS p ON i.UPC = p.UPC WHERE i.current_quantity <= i.reorder_threshold;	Inventory, store, product table 3개를 join하였다. Inventory table은 실제 수량과 threshold 및 부족 시 reorder할 수량에 대한 정보를 담고 있기 때문에 포함시키고, 어떤 매장인지 알기 위해서 store도 join한다. 또한, 그리고 inventory의 UPC 정보로만은 상품의 이름에 대해 알기 어렵기 때문에 이 또한 조인한다. 필자가 경영 1전공생이기에 비즈니스 프로세스 관리라는 수업에서 들은 바로는 reorder하는 수량은 threshold(임계치) – current(현재 수량)이 아니라 그 이상이어야 한다. 그렇기에 reorder 수량을 이렇게 설정했고, query 사용자 입장에서 단순히 얼마만큼 있어서 부족한지 뿐만 아니라 얼만큼 주문해야 하는지도 알아야 하기 때문에 이 또한 쿼리에 나타나게 했다.

```
Select: 5
------- TYPE 5 -------
**  Which products in each store are below the reorder threshold and need restocking? **
store_ID                store_name              product_name            current_quantity        reorder_threshold       reorder_quantity
S002                    CU Sinchon              Honey Butter Chips      8                       30                      120
S005                    CU Mangwon              Buldak Ramyun           3                       15                      75
```

6) "List the top 3 items that loyalty program customers typically purchase with coffee."

로열티 프로그램 고객이 커피와 함께 주로 구매하는 상위 3개 품목은?

해당 product 테이블에는 coffee라는 category가 없다. 그렇기에 product 테이블에 있는 3개의 항목 'Americano Coffee', 'Cafe Latte', 'Iced Coffee'의 이름을 query에 직접 넣어서 작성해보기로 했다.

해당 쿼리를 실행하기 위해 option 6을 선택한다면 새로운 인터페이스가 나온다. 1, 2, 3, 0 중의 option을 택할 수 있는데 만약 1이라면 americano coffee, 2라면 café latte, 3이라면 iced coffee, 0이라면 나갈 수 있다.

SELECT
  p.UPC,
  p.name AS product_name,
  p.brand,
  COUNT(*) AS purchase_count
FROM sales_transaction st1
JOIN transaction_product tp1 ON st1.transaction_ID = tp1.transaction_ID
JOIN product p1 ON tp1.UPC = p1.UPC
JOIN transaction_product tp2 ON st1.transaction_ID = tp2.transaction_ID
JOIN product p ON tp2.UPC = p.UPC
WHERE st1.loyalty_ID IS NOT NULL
  AND p1.name = 'Americano Coffee'
  AND tp2.UPC != tp1.UPC
GROUP BY p.UPC, p.name, p.brand
ORDER BY purchase_count DESC
LIMIT 3;
	Sales_transaction 1개, transaction_product 2개, product 2개로 총 5개의 테이블을 join한다.
	Tp1, p1은 커피 제품 구매 정보를 가져오기 위한 것이며, tp2, p2는 해당 거래에서 구매한 다른 상품들에 대한 정보를 가져오기 위해서이다.
	이후 제약조건들에 대해 설명해보자면, 총 3가지 조건이 AND으로 걸려있다.
	우선 loyalty_ID에 대하여 IS NOT NULL 조건을 걸어 로열티(VIP) 프로그램에 등록된 고객들에 대해서만 분석이 되도록 했다. 또한, p1.name = 'Americano Coffee' 조건을 통해 해당 쿼리를 통해 알아보려는 제품에 대해서만 조회되도록 쿼리에 넣는다. 그렇기에 이는 선택한 제품에 따라 달라지는 조건이다. 마지막으로 tp2.UPC != tp1.UPC 라는 조건을 걸었는데, 해당 제품과 함께 주로 구매하는 항목에 대해 조회해야 하기 때문에 커피 제품은 제외시킨 것이다.
	이후 같은 상품별로 그룹화하고 상위3개만 출력한다.
	다른 커피 제품에 대해서는 p1.name 일치 조건만 바꿔 똑같이 적용해주면 된다.

```
------- TYPE 6 -------
** List the top 3 items that loyalty program customers typically purchase with coffee. **
There are 3 products you can check. Select options from 1 to 3 or 0 to exit
1: Americano Coffee
2: Cafe Latte
3: Iced Coffee
0: exit from this query

Select: 1
Americano Coffee
UPC                 product_name            brand               purchase_count
1000000000013       Donut Chocolate         Dunkin              11
1000000000004       Honey Butter Chips      Haitai              9
1000000000012       Croissant               Paris Baguette      6
```

```
------- TYPE 6 -------
** List the top 3 items that loyalty program customers typically purchase with coffee. **
There are 3 products you can check. Select options from 1 to 3 or 0 to exit
1: Americano Coffee
2: Cafe Latte
3: Iced Coffee
0: exit from this query

Select: 2
Cafe Latte
UPC                  product_name          brand                 purchase_count
1000000000004        Honey Butter Chips    Haitai                9
1000000000008        Chocolate Cookies     Orion                 7
1000000000013        Donut Chocolate       Dunkin                5
```

```
------- TYPE 6 -------
** List the top 3 items that loyalty program customers typically purchase with coffee. **
There are 3 products you can check. Select options from 1 to 3 or 0 to exit
1: Americano Coffee
2: Cafe Latte
3: Iced Coffee
0: exit from this query

Select: 3
Iced Coffee
UPC                  product_name          brand                 purchase_count
1000000000012        Croissant             Paris Baguette        9
1000000000004        Honey Butter Chips    Haitai                8
1000000000008        Chocolate Cookies     Orion                 6
```

7) "Among franchise-owned stores, which one offers the widest variety of products, and how does that compare to corporate-owned stores?" 프랜차이즈와 직영 매장 중 가장 다양한 제품을 제공하는 곳은?

```
(SELECT
        ownership_type,
    s.store_ID,
    s.name AS store_name,
    COUNT(i.UPC) AS product_count
FROM store s
LEFT JOIN inventory i ON s.store_ID = i.store_ID
WHERE s.ownership_type = 'franchise'
GROUP BY s.store_ID, s.name)

UNION ALL

(SELECT
        ownership_type,
    s.store_ID,
    s.name AS store_name,
    COUNT(i.UPC) AS product_count
FROM store s
LEFT JOIN inventory i ON s.store_ID = i.store_ID
WHERE s.ownership_type = 'corporate'
GROUP BY s.store_ID, s.name
);
```

Store와 inventory 테이블을 조인하는 있어 이를 LEFT join을 한다. 이는 재고가 없는 매장도 포함시키기 위해서이다. es.ownership_type = 'franchise' 조건을 통해 프랜차이즈점만 선택한다. GROUP BY 문을 이용하여 같은 매장의 여러 제품들을 묶어 개수를 셀 수 있도록 한다. COUNT(i.UPC)를 통해 보유하고 있는 제품 개수를 세고 이를 내림차순으로 정렬하여 1등만 선택한다. 'corporate' 같은 경우에도 똑같이 해주고 이 두 결과를 UNION하여 비교한다.
그 결과 직영점이 프랜차이즈보다 더 많은 다양한 제품을 제공한다는 것을 발견할 수 있었다.

```
Select: 7
------- 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          store_ID                store_name              product_count
franchise               S002                    CU Sinchon              12
corporate               S001                    GS25 Hongik Univ        23
```