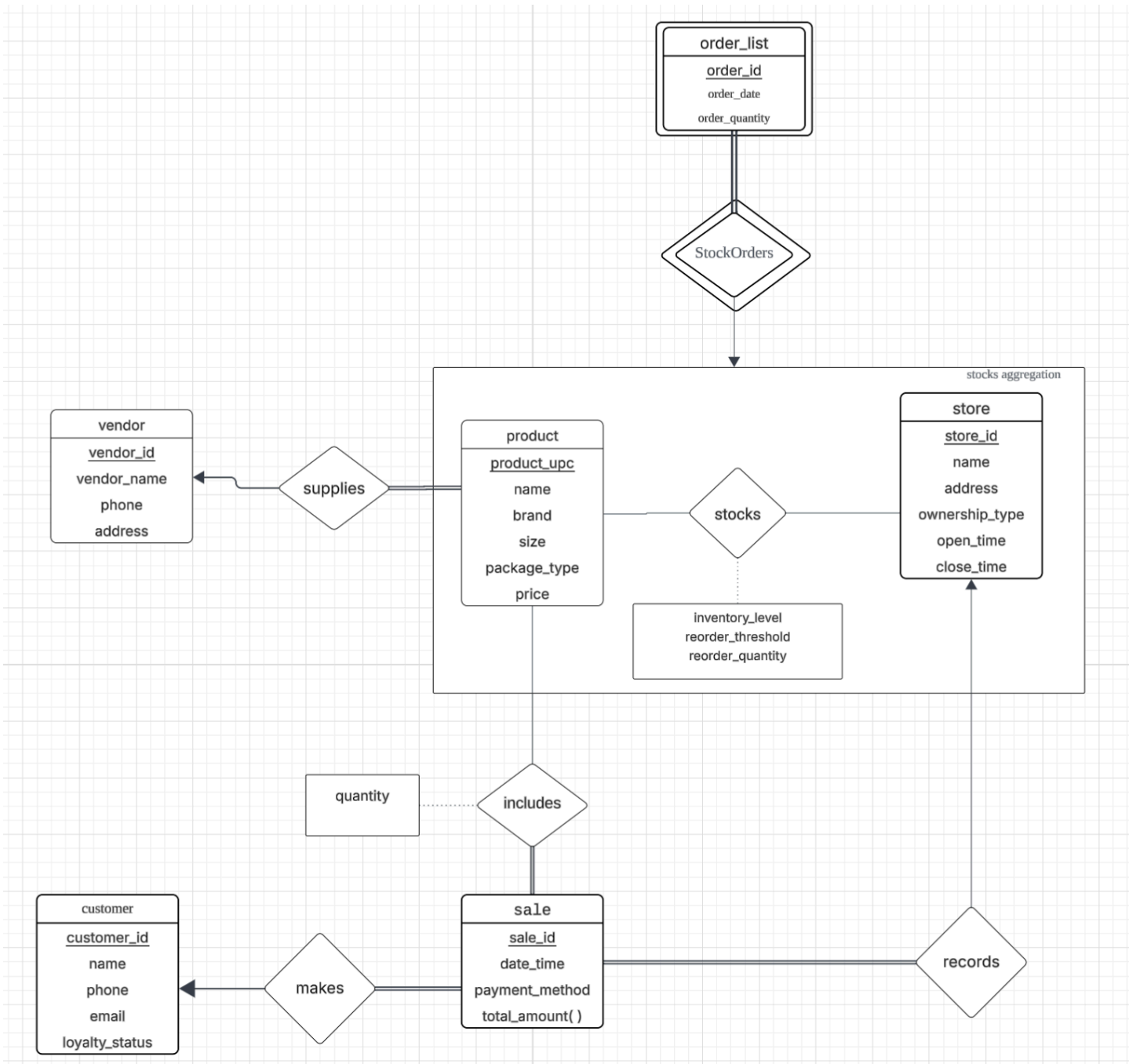


Report

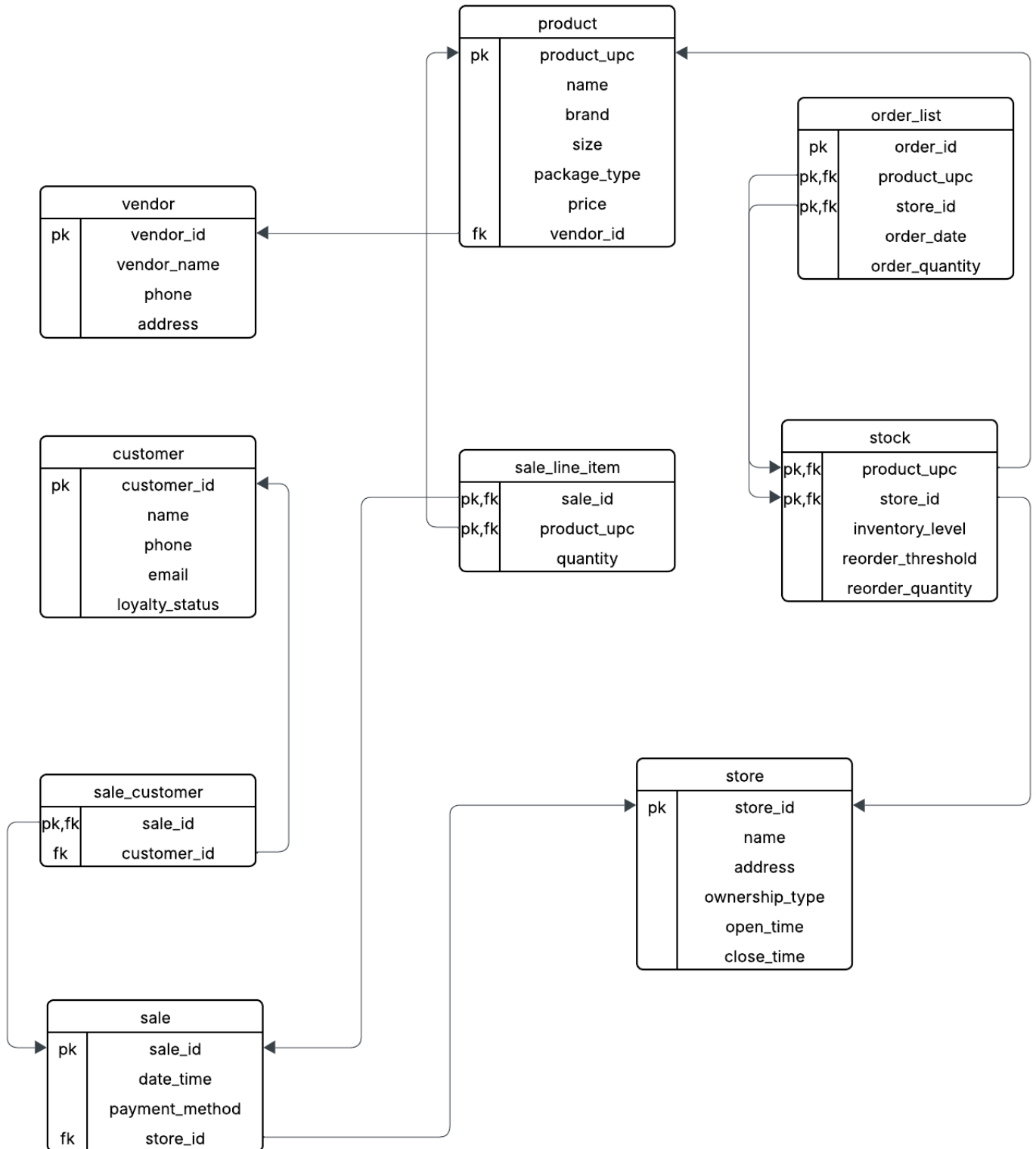
20190094_이지인

6.1 Logical Schema Design (25%)

[er diagram]



[logical schema]



• Detailed explanation of ERD to relational schema transformation • Justification for design decisions

0. E-R 모형에는 strong entity가 5개(vendor·product·store·customer·sale) 포함되어 있다. strong entity는 적합한 환원 규칙에 따라, reduction시, 같은 이름의 릴레이션을 생성하였다.

아래에는 strong entity 가 아닌, 관계와 약한 엔티티의 환원에 대해 설명해보겠다.

1. supplies 관계 환원

Vendor와 Product는 1:N 관계이며 Product는 반드시 하나의 Vendor에 의해 공급되므로 total 참여이다. 이 때문에 별도의 릴레이션을 생성하지 않고 Product 릴레이션에 vendor_id FK 칼럼을 추가하면 될 것이다. product_upc는 그대로 기본키로 유지된다.

2. records 관계 환원

Store와 Sale은 1:N 관계이며 Sale은 반드시 하나의 Store에서 발생해야 하므로 total 참여이고 Store는 판매 기록이 없을 수 있어 partial 참여이다. 따라서 별도 릴레이션 없이 Sale 릴레이션에 store_id FK 칼럼을 추가한다. 또한, sale_id는 기존과 같이 기본키로 유지된다.

3. makes 관계 환원

Customer와 Sale은 1:N 관계이나 비회원 판매를 허용하므로 Sale은 customer_id가 NULL일 수 있어 partial 참여로 모델링된다. 또한, Customer이긴 하나, 아직 구매 내역은 없을 수 있어, Customer 또한 해당 관계에서 Partial 참여에 해당한다. reduction시, 양쪽 다 partial 참여에 해당하는 1 : N 관계는 별도의 테이블을 만들어야 한다. 해당 reduction의 결과로 나온 릴레이션의 명칭은, sale_customer로 지칭하였고, many-side에 해당하는 sale_id를 pk로 설정하고, customer 테이블에서 customer_id 를 fk로 참조할 수 있도록 하고, store 테이블에서, store_id를 fk로 참조할 수 있도록 하였다.

4. includes 관계 환원

Sale과 Product는 M:N 관계이며 Sale은 최소 하나의 상품을 포함(total), Product는 아직 판매되지 않을 수도 있어 partial이다. 이 경우 양쪽 다 many side에 해당하므로, reduction시, saleLineItem 교차 릴레이션을 생성하고, PK를 (sale_id, product_upc)로 정의하였다.(양쪽의 pk의 조합이 pk가 됨) 또한, 해당 관계에 추가적인 속성으로 존재하는 quantity를 별도의 컬럼으로 포함하였다.

5. stocks 관계 환원

Store와 Product는 M:N 관계이며 양쪽 모두 partial 참여가 가능하다. 즉, store는 아직 제품이 없을 수도 있고, product 또한, 아직 어떠한 store에 입점이 되지 않았을 수도 있다. 4번의 includes 관계 환원과 마찬가지로, m:n 관계에 해당하므로, 별도의 Stock 교차 릴레이션을 생성하고 PK를 (store_id, product_upc)로 설정한다. 이때, 해당 관계에 추가적인 속성으로 존재하는, . inventory_level, reorder_threshold, reorder_quantity를 별도의 컬럼으로 포함한다.

6. order_list 집계/약한 엔티티 환원

5번의 Stock M:N 관계의 aggregation과, order_list간의 StockOrders 관계는 identifying relationship에 해당한다. 이때, order_list 쪽이 weak entity set에 해당하고, 해당 집계가 strong entity set에 해당하였다. 이때, 이러한 identifying relation은, 별도의 릴레이션을 생성하지 않는다. 그대신, weak entity set 에, weak entity set 자체의 discriminator와 더불어, identifying entity set의 pk가 weak entity set 의 pk의 일부로 추가되는 것이 알맞은 환원에 해당할 것이다. 그렇기에, strong entity set에 해당하는 집계의 pk에 해당하는, (store_id, product_upc) PK를 OrderList 릴레이션의 pk 집합의 구성요소로 추가하여, (order_id, store_id, product_upc) 복합키로 OrderList 릴레이션의 pk를 설정하였다.

6.2 Normalization Analysis (20%)

[logical schema이자, bcnf 만족 스키마]

```
erDiagram
    vendor ||--o{ product : "has"
    product ||--o{ order_list : "has"
    product ||--o{ stock : "has"
    customer ||--o{ sale : "has"
    sale ||--o{ sale_customer : "has"
    sale ||--o{ store : "has"
    order_list ||--o{ stock : "has"

    vendor {
        string vendor_id PK
        string vendor_name
        string phone
        string address
    }
    product {
        string product_upc PK
        string name
        string brand
        string size
        string package_type
        float price
        string vendor_id FK
    }
    order_list {
        string order_id PK
        string product_upc PK_FK
        string store_id PK_FK
        string order_date
        float order_quantity
    }
    stock {
        string product_upc PK_FK
        string store_id PK_FK
        float inventory_level
        float reorder_threshold
        float reorder_quantity
    }
    customer {
        string customer_id PK
        string name
        string phone
        string email
        string loyalty_status
    }
    sale {
        string sale_id PK
        string date_time
        string payment_method
        string store_id FK
    }
    sale_customer {
        string sale_id PK_FK
        string customer_id FK
    }
    store {
        string store_id PK
        string name
        string address
        string ownership_type
        string open_time
        string close_time
    }
```

1. 스키마 → vendor(vendor_id, vendor_name, phone, address)
2. 기본키(PK) → vendor_id
3. 기본키 이외의 후보키 → vendor_name
4. 비자명 FD① → vendor_id → vendor_name, phone, address
5. 비자명 FD② → vendor_name → vendor_id, phone, address
6. 좌변 두 개 모두 후보키에 해당하므로, 모두 슈퍼키 조건을 충족한다.

② product → BCNF 만족

1. 스키마 → product(product_upc, name, brand, size, package_type, price, vendor_id)
2. PK → product_upc
3. 비자명 FD → product_upc → name, brand, size, package_type, price, vendor_id
4. 공급사 하나가 여러 상품을 갖기 때문에 vendor_id → ~ 는 성립하지 않는, fd이다.(즉, 해당 릴레이션에 이러한 fd는 존재하지 않음)
5. product_upc⁺ 가 모든 속성을 포함하기 때문에, 좌변이 슈퍼키이다.

③ store → BCNF 만족

1. 스키마 → store(store_id, name, address, ownership_type, open_time, close_time)
2. PK → store_id, pk 이외의 후보키 → name (vendor와 마찬가지로, 동일한 이름을 지닌 store는 없음을 가정하고 있기 때문에, 이 또한 후보키이다.)
3. FD① → store_id → name, address, ownership_type, open_time, close_time
4. FD② → name → store_id, address, ownership_type, open_time, close_time
5. 두 FD 좌변 모두 후보키에 해당한다. 때문에, 속성 폐쇄가 테이블 전체 포함한다.

④ customer → BCNF 만족

1. 스키마 → customer(customer_id, name, phone, email, loyalty_status)
2. PK → customer_id, pk 이외의 후보키 → email
3. FD① → customer_id → name, phone, email, loyalty_status
4. FD② → email → customer_id, name, phone, loyalty_status
5. 두 좌변이 모두 후보키로 슈퍼키 조건 충족하므로, 해당 릴레이션에선, 고객 정보를 중복 없이 유지하는 것이 가능하다.

⑤ sale → BCNF 만족

1. 스키마 → sale(sale_id, date_time, payment_method, store_id)
2. PK → sale_id

3. $FD \rightarrow sale_id \rightarrow date_time, payment_method, store_id$
다른 FD는 존재하지 않고, 좌변이 기본키이므로 슈퍼키에 해당한다.

⑥ $sale_customer \rightarrow BCNF$ 만족

1. 스키마 $\rightarrow sale_customer(sale_id, customer_id)$
2. $PK \rightarrow sale_id$ (1:0-1 관계), $FD \rightarrow sale_id \rightarrow customer_id$
3. $customer_id \rightarrow sale_id$ 는, 하나의 고객은 여러개의 영수증을 지닐 수 있으므로, 성립하지 않는 fd이다.
4. 좌변이 PK라서 슈퍼키 조건을 충족하게 된다.

⑦ $sale_line_item \rightarrow BCNF$ 만족

1. 스키마 $\rightarrow sale_line_item(sale_id, product_upc, quantity)$
2. $PK \rightarrow (sale_id, product_upc)$
3. $FD \rightarrow (sale_id, product_upc) \rightarrow quantity$
4. 다른 FD는 존재하지 않고, 좌변이 기본키이므로 슈퍼키에 해당한다.

⑧ $stock \rightarrow BCNF$ 만족

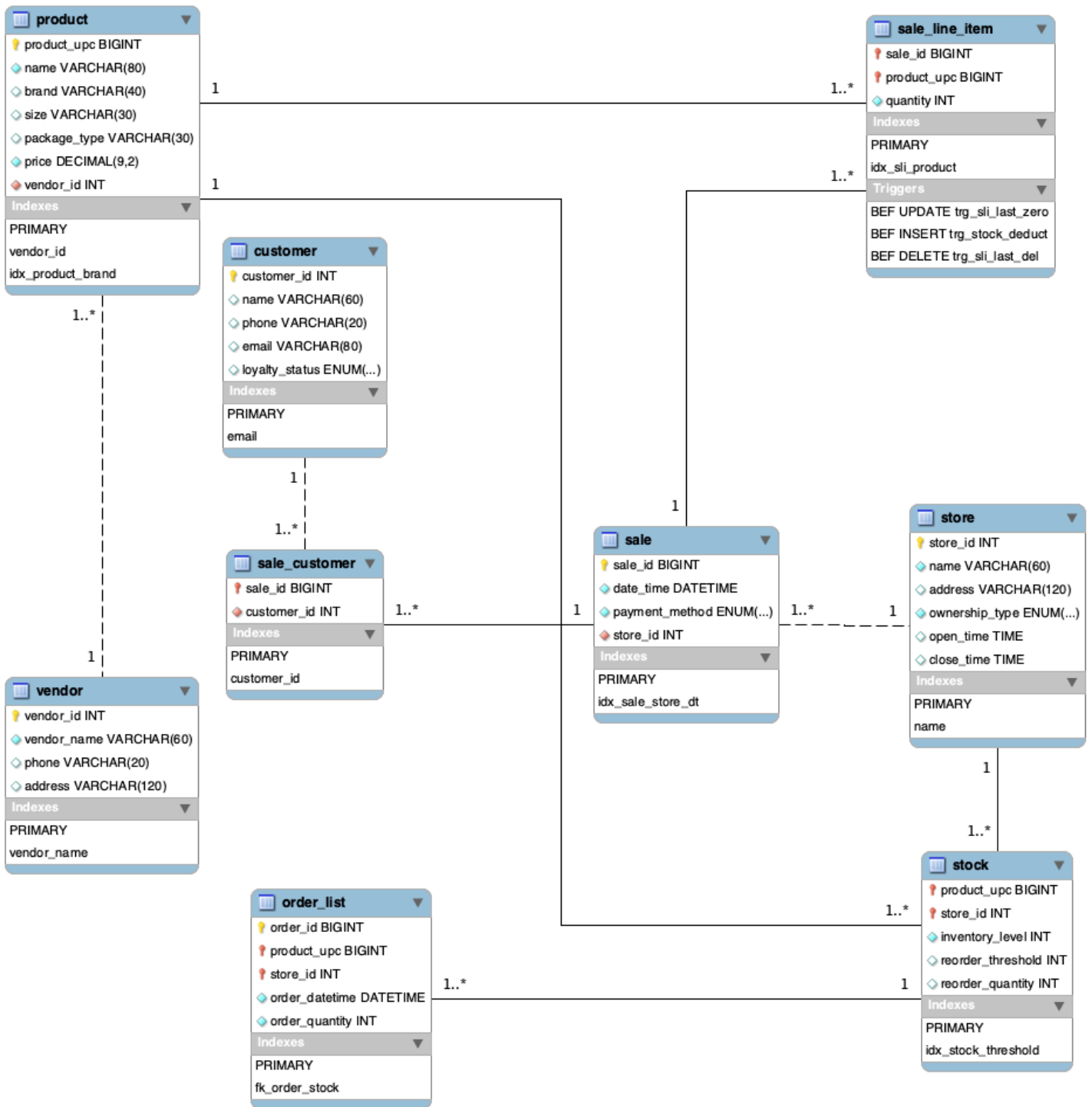
1. 스키마 $\rightarrow stock(product_upc, store_id, inventory_level, reorder_threshold, reorder_quantity)$
2. $PK \rightarrow (product_upc, store_id)$
3. $FD \rightarrow (product_upc, store_id) \rightarrow inventory_level, reorder_threshold, reorder_quantity$
4. 다른 FD는 존재하지 않고, 좌변이 기본키이므로 슈퍼키에 해당한다.

⑨ $order_list \rightarrow BCNF$ 만족

1. 스키마 $\rightarrow order_list(order_id, product_upc, store_id, order_datetime, order_quantity)$
2. $PK \rightarrow (order_id, product_upc, store_id)$
3. 상품의 주문은, 각각의 상품이 재입고할 시점이 되면, 주문이 이루어지는 것이므로, 각 점포의 각 상품별로, 주문 시각은 상이하다. 따라서, $(order_id, store_id)$ 만으로는 시간 결정 불가하므로, 이는 해당 릴레이션의 fd가 아니다.
4. 해당 릴레이션의 유일한 fd는, $(order_id, product_upc, store_id) \rightarrow order_datetime, order_quantity$ 이며, 좌변이 기본키이므로 슈퍼키에 해당한다.

6.3 Physical Implementation (15%)

[physical_digram]



• Data type selection rationale • Constraint implementation and business rule enforcement • sample data description

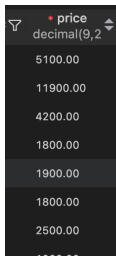
6.3.1 데이터 타입 선정 근거

점포·공급사·고객 식별자(store_id,vendor_id,customer_id)는 20억 건 정도의 처리가 가능한 INT로 설정하였다. 하지만, product_upc와 sale_id는 BIGINT UNSIGNED로 선언하였는데, 그러한 이유는, 이 둘은 모든 전역에서 식별자로 쓰이고, vendor_id와, customer_id등의 수보다, 제품의 바코드 수나, 각 점포별 거래내역을 전역적으로 식별자로 관리하려면, 훨씬 큰 범위의 수가 필요하기 때문이다.

상품명, 브랜드, 주소 등 가변 길이 문자열은 VARCHAR(40 ~ 120)로 두어 실제 길이에 비례해 공간을 사용하게 하였다.

또한, ownership_type,payment_method,loyalty_status는, 종류가 상품명, 브랜드, 주소처럼, 무한하기 보다, 몇 가지 종류로 한정적이기 때문에, 카테고리처럼 관리할 수 있는, ENUM으로 고정 목록을 강제하였다.

또한, 금액과 관련된 정보는 통화 단위(원)로 표시가 가능하기 위해, 비교적 긴 DECIMAL(9,2)로 선언하였다. 때문에,



price decimal(9,2)
5100.00
11900.00
4200.00
1800.00
1900.00
1800.00
2500.00

이렇게 저장된 가격을 보면 바로 5100원이군, 11900원이군 이렇게 파악할 수 있도록 하였다.

다음은 시간과 관련된 속성 타입에 대해 이야기하겠다. 먼저 store의 영업 시각(open_time,close_time)은 날짜와 무관하므로 TIME을 사용하였다. 즉, 아래와 같이 00:00:00 / 00:00:00이면, 24시간 영업으로 파악할 수 있고, 07:00:00/02:00:00이면, 오전 7시 오픈 ~ 오전 2시 문 닫음으로 파악할 수 있을 것이다.

	store_id int	name varchar(60)	address varchar(120)	ownership_type enum('FRANCHISE','CORPORATE')	open_time time	close_time time
>	101	강남역점	서울 강남구 테헤란로 1	FRANCHISE	00:00:00	00:00:00
>	102	홍대입구점	서울 마포구 와우산로 2	CORPORATE	07:00:00	02:00:00
>	103	잠실새내점	서울 송파구 올림픽로 3	FRANCHISE	06:00:00	24:00:00
>	104	종로3가점	서울 종로구 돈화문로 4	CORPORATE	00:00:00	00:00:00
>	105	부산서면점	부산 진구 중앙대로 5	FRANCHISE	00:00:00	00:00:00

이것 말고, sale 테이블의 판매 시간이나, order_list 테이블의 주문 시간은 날짜와도 관련이 있기 때문에, DATETIME로 지정하였다.

재고량·수량·임계치 등 모든 산술적인 속성은 INT와 CHECK 제약을 조합해 음수 입력을 방지하였다. 음수 입력을 방지하였기 때문에, 애플리케이션 단에서 사용자가 불필요한 검사를 반복 수행하지 않아도 데이터베이스 스스로 비정상 값을 거절할 수 있게 된 것이다.

```
CREATE TABLE stock (  
  product_upc      BIGINT UNSIGNED,  
  store_id         INT,  
  inventory_level  INT NOT NULL CHECK (inventory_level >= 0),  
  reorder_threshold INT NULL CHECK (reorder_threshold >= 0),  
  reorder_quantity INT NULL CHECK (reorder_quantity > 0),
```

6.3.2 제약 조건 및 비즈니스 규칙

1) pk, unique constraint

모든 엔터티 테이블은 행을 유일하게 식별할 수 있도록 하기 위해 단일 속성을 PK로 지정하였다.

```
CREATE TABLE vendor ( vendor_id INT PRIMARY KEY ... );
```

```
CREATE TABLE product ( product_upc BIGINT UNSIGNED PRIMARY KEY ... );
```

```
CREATE TABLE store ( store_id INT PRIMARY KEY ... );
```

```
CREATE TABLE customer( customer_id INT PRIMARY KEY ... );
```


CREATE TABLE sale (sale_id BIGINT PRIMARY KEY ...);

vendor_name·store.name·customer.email에는 UNIQUE 키를 추가하여(vendor_name VARCHAR(60) NOT NULL UNIQUE) 현실 세계에서 중복이 허용되지 않는 것이 훨씬 더 자연스러운 브랜드, 점포, 이메일의 유일성을 데이터베이스 레벨에서 보장할 수 있도록 했다.

엔티티 테이블이 아닌, 복합관계에서 파생된 테이블은 관계의 두 PK를 그대로 합친 복합 PK를 사용해 중복 행이 발생하지 않도록 하였다.

PRIMARY KEY(product_upc, store_id) ← stock

PRIMARY KEY(order_id, product_upc, store_id) ← order_list

PRIMARY KEY(sale_id, product_upc) ← sale_line_item

때문에, 동일 (store,product) 재고를 두 번 INSERT 하려 시도하는 등의 시도를 하면, db에서 오류를 반환해서 이러한 작업을 막아줄 수 있게 되었다.

2) fk + CASCADE/RESTRICT

모든 FK는 각각의 상황에 맞게, ON UPDATE CASCADE와 ON DELETE 동작을 다르게 지정하였다.

관계	SQL 정의	비즈니스적인 의미
product → vendor	FOREIGN KEY (vendor_id) REFERENCES vendor(vendor_id)(default RESTRICT)	공급사가 먼저 존재해야 상품을 등록할 수 있다. 그렇기 때문에, 공급사 삭제는 판매 상품이 남아 있을 때 삭제할 수 없도록 한다.
sale → store	ON UPDATE CASCADE ON DELETE RESTRICT	store의 코드가 변경되면 과거 매출도 자동 갱신(연쇄 갱신)될 수 있도록 하였다. 그러나 store를 삭제할 때는 매출 이력을 보존하기 위해 거부하도록 하였다.
sale_customer	sale_id ON DELETE CASCADE / customer_id ON DELETE RESTRICT	영수증(sale)이 삭제되면 연결된 sale_customer의 레코드도 함께 삭제될 수 있도록 하였다. 반대로 고객을 삭제하면 default 값이 적용되어, 과거 영수증의 FK를 NULL 로 두지 말고 삭제 자체를 차단할 수 있도록 해 구매 이력을 보호할 수 있다.
stock, order_list	(product_upc,store_id) ON DELETE CASCADE	점포가 폐점되면 해당 점포의 재고·발주 이력은 자동적으로 정리할 수 있도록 하였다.

위의 적용을 설명해보자면, 영구적으로 보존이 필요한 엔터티(sale,product)에는 RESTRICT을, 종속적·소모성 데이터에 해당하는, stock,order_list에는 CASCADE를 두어, 데이터 수명 주기를 관리할 수 있도록 했다.

3) CHECK 제약 — 도메인 무결성

각 테이블에 알맞은 값의 범위를 강제하는 CHECK를 추가해 도메인 무결성을 보장하고자 했다.

```
price          DECIMAL(9,2) CHECK(price > 0)
```

```
inventory_level INT CHECK(inventory_level >= 0)
```

```
reorder_quantity INT CHECK(reorder_quantity > 0)
```

```
quantity       INT CHECK(quantity > 0)
```

이뿐만 아니라, 모든 날짜와 시간 필드는 DATETIME/TIME의 고유한 format을 사용해 잘못된 날짜 입력을 MySQL 파싱 단계에서 차단할 수 있다.

4) 비즈니스 로직 trigger

SQL만으로 표현하기 어려운 중복적인 절차와 규칙은 트리거를 통해 보완해보았다.

트리거	위치 / 시점	핵심 코드	강제 규칙
trg_stock_deduct	BEFORE INSERT ON sale_line_item	① 해당 점포 재고 잠금 ② cur_stock < NEW.quantity → SIGNAL '재고 부족' ③ 재고 차감을 UPDATE	판매 시점에 실재고 미만은 판매를 금지하고, 재고를 실시간 동기화한다.
trg_sli_last_del	BEFORE DELETE ON sale_line_item	삭제 전에, 동일 영수증 라인 수를 COUNT하고, → 1 이면 SIGNAL을 보낸다.	영수증이 최소 1 개 품목을 유지할 수 있도록 한다.
trg_sli_last_zero	BEFORE UPDATE ON sale_line_item	만약 하나의 sale line이라면, 이것의 수량이 quantity<=0 으로 업데이트 되는 것을 차단했다.	UPDATE에서도 trg_sli_last_del에 서와 같은 동일 규칙 유지할 수 있도록 한다.

```
4517 17:49:03 INSERT INTO sale_line_item (sale_id, product_upc, quantity) VALUES (10001, 8801111000002, 9999)
45... 17:49:03 UPDATE sale_line_item SET quantity = 0 WHERE sale_id = 10060 AND product_upc = 8809999000002
```

Error Code: 1644. 재고 부족
Error Code: 1644. Sale 은 최소 1개의 line item 을 가져야...

위의 트리거로 인해 위의 스크린샷에서와 같이, 재고가 부족한데, 이를 판매 아이템으로 추가하려고 하면, 재고 부족 오류가 뜨고, 특정 sale의 유일한 sale line item에 해당하는 것의 수량을 0으로 바꾸려고하면, 위와 같은 오류가 나도록 하였다.

5) 보조 인덱스

업무 쿼리를 분석해 카디널리티가 높고, WHERE·JOIN 조건에 자주 등장하는 열에 인덱스를 추가했다.

```
CREATE INDEX idx_product_brand ON product(brand); -- 브랜드 검색
```

```
CREATE INDEX idx_sale_store_dt ON sale(store_id, date_time); -- 점포+기간 매출
```

CREATE INDEX idx_stock_threshold ON stock(store_id,inventory_level,reorder_threshold);

CREATE INDEX idx_sli_product ON sale_line_item(product_upc); -- 상품 판매량 집계

6) Cardinality & Participation 구현 세부 설명

논리적 변환 규칙을 실제 DDL 수준에서 어떻게 실현했는지를 관계(relationship)별로 정리해보았다.

관계 이름	변환 후 테이블	카디널리티 구현	참여 제약 구현	SQL
supplies (Vendor 1 : N Product)	product	N-측(product) PK 안에 1-측 FK를 둬으로써 1:다를 구현했다.	Product가 total 참여이므로 FK에 NOT NULL을 선언해 “공급사 없는 상품”을 금지하였다. Vendor는 partial이므로 Vendor 테이블에는 별도 제약이 없다.	vendor_id INT NOT NULL, FOREIGN KEY (vendor_id)
records (Store 1 : N Sale)	sale	다수의 Sale 행이 하나의 Store를 가리키도록 store_id FK 를 many side에 배치했다.	Sale이 total 참여이므로 FK에 NOT NULL + ON DELETE RESTRICT을 적용했다.	store_id INT NOT NULL ... ON DELETE RESTRICT
makes (Customer 0..1 : N Sale)	sale_customer	양쪽이 partial이므로 별도 릴레이션이 필요했다. many side에 해당하는 PK를 sale_id 단독으로 두어 한 영수증이 최대 한 고객만 연결될 수 있도록 했다.	Customer 측은 참여가 선택적이다. 즉, 모든 고객이 구매 이력이 있는 것은 아니다.	PRIMARY KEY(sale_id)...ON DELETE CASCADE ...RESTRICT
includes (Sale M : N Product)	sale_line_item	M:N를 (sale_id, product_upc)로 복합 PK로 표현하였다. many to many의 total 참여를(sale은, 판매내역이므로, sale_line_item 무조건 있어야만 한다.) 이를 4) 비즈니스 로직 trigger에서 언급한 trg_sli_last_del와, trg_sli_last_zero로 최대한 강제하고자 했다.	Sale 은 total, Product 는 partial.	PRIMARY KEY(sale_id, product_upc)...

stocks (Store M : N Product)	stock	<div> 동일 패턴으로 복합 PK(store_id, product_upc). 각 점포-상품 쌍은 하나의 재고 레코드만 가짐. </div>	<div> 양쪽 모두 partial이라 FK를 NOT NULL로 두지는 않았지만, 부모 삭제 시 CASCADE로 자동 정리했다. </div>	ON DELETE CASCADE
reorder/aggregation (order_list ↔ stock)	order_list	<div> 약한 엔티티의 PK = order_id + product_upc) → (M:N 집계된 Stock 쪽과 1:N. </div>	<div> order_datetime·order_quantity NOT NULL로 “발주 시각·수량 없음”을 금지하였다. </div>	PRIMARY KEY(order_id, product_upc, store_id)

6.3.3 샘플 데이터 구성

샘플 데이터는 현실성을 최우선으로, “카테고리적인 다양성과, 시계열 분포와, 용량 테스트가 가능하도록 여러 데이터를 포함시켰다. 공급사는 식품사와 음료사를 합쳐 14개를 만들었고, 상품은 과자·라면·생활용품·커피 등 41종으로 확장했다. 10개 점포는 FRANCHISE 6 : CORPORATE 4의 구조로 배치해 7번째의 쿼리(소유형태를 비교하는 쿼리에 해당)이 의미 있는 결과를 반환하도록 하였다.

판매 데이터는 60건 정도로 삽입하였고, 최근 20일간 고르게 분포시켜 월간·분기별 집계를 담당하는 쿼리가 집계 오류 없이 잘 동작함을 확인할 수 있었다. 또한, 각 영수증에 1~3개 품목을 섞어 sale_line_item을 120건 정도로 생성하였다. 재고 테이블에 해당하는 stock에는 “상품 41 × 점포 10”를, 기본으로 생성하되, 일부 상품·점포 조합을 의도적으로 누락시켜 ‘미입점’의 시나리오도 가능하게 해, 7번째의 쿼리(소유형태를 비교하는 쿼리에 해당)이 의미 있는 결과를 반환하도록 하였다.

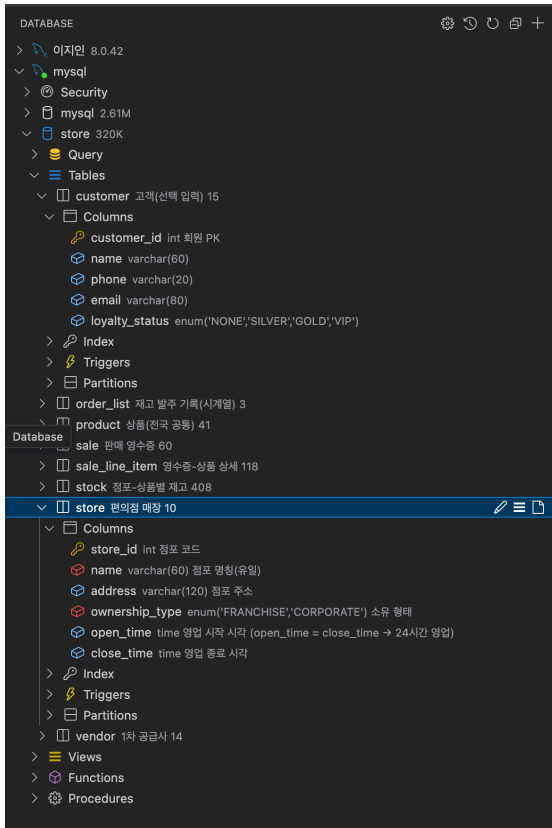
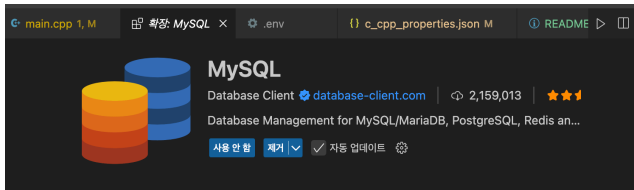
마지막으로 order_list에 발주 이력 3건을 추가하였다.

6.4 Application Development (30%)

- Database connectivity implementation details • Query implementation • Error handling and user interface design
- 6.4.1 데이터베이스 연결
- 애플리케이션은 호환성과 배포 편의성을 위해 MySQL C API 하나만 사용한다. mysql_init() 이후 mysql_real_connect()로 단일 세션을 만들고, 종료 시 mysql_close()를 호출하는 전형적 패턴을 사용하였다.

예외 처리 함수에 해당하는 `die()`를 통해, 오류 메시지를 출력하고, 메모리 자원을 회수하고 그 이후 프로세스가 종료될 수 있도록 해서, 메모리 누수 등이 남지 않도록 하였다.

데이터베이스 연결은, 가장 기본적인 아래의 `extension`을 사용했다.



6.4.2 Query Implementation

① 상품 재고 현황 (Product Availability)

첫 번째 쿼리는 사용자가 입력한 UPC 코드 또는 키워드를 기반으로 전국 매장 재고를 일람하는 기능을 수행한다. `stock-store-product` 세 테이블을 조인하고, `WHERE ... OR ... OR ...` 형식으로 UPC·상품명·브랜드 모두에 패턴 매칭을 적용하였다.

UPC는 기본키에 해당하므로, 정확하게 일치 가능성이 있지만, 키워드 검색은 `LIKE '%키워드%'`를 사용해 부분 일치를 허용했고, `concat`으로, 문자열과 `key` 값을 이어붙였다. `select`를 통해 출력되는 컬럼은 점포 식별자·점포명·UPC·상품명·재고량 다섯 가지에 해당하며, 최종적인 출력은 점포 번호 기준으로 정렬해 동일 상품을 여러 점포에서 한눈에 비교할 수 있게 하였다.

예시 1처럼 UPC만 주어질 때는 대상 상품 하나만, 예시 2처럼 ‘콩콩’ 같이 광범위한 키워드를 주면 수십 행이 반환되어 검색 폭을 넓힐 수 있는 것을 확인할 수 있다.

- 예시 1

1 UPC / 키워드 입력 > 8805555000001 store_id store product_upc name inventory_level

101 강남역점 8805555000001 콩콩 아메리카노 250ml 5
108 광주상무점 8805555000001 콩콩 아메리카노 250ml 12
107 대구동성로점 8805555000001 콩콩 아메리카노 250ml 12
106 대전둔산점 8805555000001 콩콩 아메리카노 250ml 12
105 부산서면점 8805555000001 콩콩 아메리카노 250ml 29
109 수원역점 8805555000001 콩콩 아메리카노 250ml 64
110 인천송도점 8805555000001 콩콩 아메리카노 250ml 25
103 잠실새내점 8805555000001 콩콩 아메리카노 250ml 94
104 종로3가점 8805555000001 콩콩 아메리카노 250ml 76
102 홍대입구점 8805555000001 콩콩 아메리카노 250ml 63

- 예시 2

1 UPC / 키워드 입력 > 콩콩 store_id store product_upc name inventory_level

101 강남역점 8807777000002 콩콩 스위트라떼 500ml 9
101 강남역점 8807777000001 콩콩 콜드브루 500ml 53
101 강남역점 8805555000006 콩콩 헤이즐넛 250ml 7
101 강남역점 8805555000005 콩콩 모카 250ml 86
101 강남역점 8805555000004 콩콩 콜드브루 250ml 88
101 강남역점 8805555000003 콩콩 디카페인 250ml 55
101 강남역점 8805555000002 콩콩 라떼 250ml 15
101 강남역점 8805555000001 콩콩 아메리카노 250ml 5
108 광주상무점 8807777000002 콩콩 스위트라떼 500ml 47
108 광주상무점 8807777000001 콩콩 콜드브루 500ml 91
108 광주상무점 8805555000006 콩콩 헤이즐넛 250ml 57
108 광주상무점 8805555000005 콩콩 모카 250ml 52
108 광주상무점 8805555000004 콩콩 콜드브루 250ml 31
108 광주상무점 8805555000003 콩콩 디카페인 250ml 52
108 광주상무점 8805555000002 콩콩 라떼 250ml 71

108 광주상무점 8805555000001 콩콩 아메리카노 250ml 12

..생략

106 대전둔산점 8805555000001 콩콩 아메리카노 250ml 12

105 부산서면점 8807777000002 콩콩 스위트라떼 500ml 29

105 부산서면점 8807777000001 콩콩 콜드브루 500ml 89

105 부산서면점 8805555000006 콩콩 헤이즐넛 250ml 19

105 부산서면점 8805555000005 콩콩 모카 250ml 75

105 부산서면점 8805555000004 콩콩 콜드브루 250ml 100

105 부산서면점 8805555000003 콩콩 디카페인 250ml 14

105 부산서면점 8805555000002 콩콩 라떼 250ml 79

105 부산서면점 8805555000001 콩콩 아메리카노 250ml 29

109 수원역점 8807777000002 콩콩 스위트라떼 500ml 90

109 수원역점 8807777000001 콩콩 콜드브루 500ml 8

... 생략

102 홍대입구점 8805555000004 콩콩 콜드브루 250ml 90

102 홍대입구점 8805555000003 콩콩 디카페인 250ml 63

102 홍대입구점 8805555000002 콩콩 라떼 250ml 11

102 홍대입구점 8805555000001 콩콩 아메리카노 250ml 63

② 지난 한 달 베스트셀러 (Top-Selling Items)

```
===== Convenience-Store Queries =====
1.Product Availability
2.Top-Selling Items (1M)
3.Store Performance (Q)
4.Vendor Statistics
5.Reorder Alert
6.Coffee Bundles
7.Variety by Ownership
0.Exit
> 2
store_id  store                product_upc          name                revenue
101      강남역점            8809999000002      마운틴 라떼 350ml    19200.00
102      홍대입구점          8811111000005      씨티카페 콜드브루 500ml14400.00
103      잠실새내점          8808888000004      데일리빈 카라멜마끼아또 350ml26400.00
104      종로3가점           8801046290460      울삼푸 교복워시 1L    25500.00
105      부산서면점          8801051034851      피지 프로클린젤 2.7L  47600.00
106      대전둔산점          8801051034851      피지 프로클린젤 2.7L  59500.00
107      대구동성로점        8809107600814      슈가버블 세탁세제 2.4L44500.00
108      광주상무점          8809107600814      슈가버블 세탁세제 2.4L35600.00
109      수원역점            8801051034851      피지 프로클린젤 2.7L  59500.00
110      인천송도점          8811111000003      씨티카페 아메리카노 350ml33000.00
```

두 번째 쿼리는 최근 1 개월 동안 각 점포에서 가장 많이 팔린 단일 상품을 찾는다.

먼저 sale_line_item·sale·product·store를 조인한 뒤 WHERE s.date_time >= CURRENT_DATE - INTERVAL 1 MONTH 조건으로 기간을 한정하였다. 내부 with 절 product_sales에서 SUM(quantity × price)로 매출액을 계산하고 점포·상품 단위로 그룹화하였다. 이후 ROW_NUMBER() OVER(PARTITION BY store_id ORDER BY revenue DESC) 윈도우 함수를 사용하여, 점포별 매출 1위에만 순위 1을 부여할 수 있도록 하였다. 내부 쿼리가 아닌, 외부 쿼리는 WHERE r=1 필터로 순위 1만 남기기 때문에, 점포당 정확히 한 행만 출력된다. 매출 계산 과정에 가격 컬럼이 포함되어 있기 때문에, 수량 기준이 아닌 금액 기준 랭킹을 제공할 수 있다. 또한, 중간에 CAST 함수도 사용해, 타입 명시/변환이 가능할 수 있도록 하였다.

③ 이번 분기 최고 매출 점포 (Store Performance)

```
> 3
```

store_id	name	revenue
105	부산서면점	169050.00

세 번째 쿼리는 현재 분기(QUARTER + YEAR) 동안 가장 많은 매출을 낸 점포를 ‘단일’ 행으로 도출하는 쿼리이다. 이 쿼리 또한, 2번 쿼리처럼, 매출 관련 쿼리이므로, 내부의 SELECT에서 SUM(quantity × price)로 매출을 합산하고 점포별로 그룹화하였다. 외부 HAVING 절 안에는 서브쿼리를 배치해 모든 점포의 매출 합계를 다시 계산한 뒤 최댓값과 비교를 수행했다. 분기를 지정하는 기간 필터에 대해 설명해보자면, QUARTER(s.date_time)=QUARTER(CURDATE()) AND YEAR(s.date_time)=YEAR(CURDATE()) 조건으로 구현하였다. 최종 결과에는 점포 ID·점포명·매출액 세 필드가 출력될 수 있도록 했다.

④ 공급사별 상품·판매 통계 (Vendor Statistics)

```
> 4
```

vendor_id	vendor_name	product_types	total_units_sold
11	콩콩커피	8	58
12	데일리빈	5	55
13	마운틴로스터스	5	30
14	씨티카페	5	50
2	오리온	4	21
1	본사	3	31
4	코카콜라	2	22
8	농심	2	26
9	빙그레	2	16
3	팔도	1	6
5	하이트진로	1	13
6	롯데칠성	1	14
7	동원	1	6
10	SPC삼립	1	14

네 번째 쿼리는 공급사를 중심으로 두 가지 지표를 동시에 계산한다.

먼저 첫 번째 지표는, ①공급사별 취급 상품 종류 수이다. 두 번째 지표는, ②누적 판매 수량이다.

이 쿼리에선, vendor를 좌측에 두고 LEFT JOIN product, 이어서 LEFT JOIN sale_line_item을 체인 방식으로 연결하였다. 판매가 전혀 없는 상품도 보여야 하기 때문에, LEFT JOIN을 선택한 것이고, CASE WHEN SUM(quantity) IS NULL THEN 0 구문으로 NULL을 0으로 변환해 통계를 일관화해서 알맞은 결과를 얻을 수 있도록 하였다. 그룹화는 공급사 PK인 vendor_id 기준으로 수행했다. 정렬은 ORDER BY product_types DESC로 취급하는 개별 상품 관리 단위(COUNT(DISTINCT p.product_upc))가 많은 순을 우선하여 자사 거래 비중을 조정할 때 ‘대형 공급사’를 곧바로 식별할 수 있게 하였다. 판매량이 적더라도 개별 상품 관리 단위가 많은 공급사가 상위에 위치할 수 있도록 했다.

⑤ 재고 부족 알림 (Inventory Alert)


```
0.Exit
> 5
```

store_id	store	product	inventory_level	reorder_threshold
101	강남역점	사과 1개	54	55
102	홍대입구점	마운틴 아메리카노 350ml	6	55
102	홍대입구점	견과류 믹스 30g	21	22
103	잠실새내점	울삼푸 교복워시 1L	12	13
104	종로3가점	콩콩 모카 250ml	20	23
104	종로3가점	참치마요 삼각김밥	72	73
105	부산서면점	사과 1개	39	43
106	대전둔산점	견과류 믹스 30g	29	33
108	광주상무점	진라면 매운맛	6	11
108	광주상무점	데일리빈 헤이즐넛 라떼 350ml	51	53
109	수원역점	피지 프로클린젤 2.7L	47	48
110	인천송도점	마운틴 라떼 350ml	3	4
110	인천송도점	씨티카페 아메리카노 350ml	14	18
110	인천송도점	씨티카페 에스프레소 60ml	42	44

다섯 번째 쿼리는 매장의 재고가 재주문 임계치보다 낮은 모든 레코드를 찾아내는 기능을 하는 쿼리다. stock을 중심으로 store·product를 조인하고, WHERE inventory_level < reorder_threshold 조건으로, 부족 재고를 판별했다. 출력 column은 점포 ID·이름, 상품명, 현재 재고량, 임계치 다섯 개로 구성했다. 정렬은 stock 순으로 진행하였다.

⑥ 일반 고객이 아닌, 프리미엄 고객의 커피와 같이 구매한 상품 번들 분석 (Customer Patterns)

```
> 6
커피 검색어 (ex. 아메리카노) > 아메리카노
item cnt
씨티카페 에스프레소 60ml 8
콩콩 헤이즐넛 250ml 4
콩콩 라떼 250ml 2
```

여섯 번째 쿼리는 특정 커피 키워드와 VIP·GOLD·SILVER 고객을 교집합으로 두고, 해당 프리미엄 고객들이, 커피와 함께 가장 많이 사 간 동반 상품 Top 3를 도출한다. 첫 내부 with 절 coffee_sales는 조건에 부합하는 영수증 번호만 추리는 역할을 수행한다. 두 번째 with에 해당하는 bundled는 이 영수증에 포함된 모든 상품을 다시 합산하되, 커피 자체는 NOT LIKE 조건으로 제외한다. 그리고, SUM(quantity)로 총 판매 수량을 집계하고 ORDER BY cnt DESC LIMIT 3으로 상위 세 품목만 남긴다. 문자열 패턴은 사용자의 입력을 받아, '%' || keyword || '%' 형태로 생성할 수 있도록 하였다.

⑦ 소유 형태별 상품 다양성 비교 (Franchise vs Corporate Variety)

마지막 쿼리는 프랜차이즈와 직영점 각각에서 가장 다양한 상품을 보유한 점포를 찾는다. 내부 서브쿼리 t가 점포별로 COUNT(DISTINCT product_upc)를 계산한다. 그 결과를 한 번 더 감싼 서브쿼리 c2는 MAX(kinds)를 통해, 소유형태별 최대값을 추출한다. 마지막 단계에서 c1(점포별 결과)과 c2(최대값)에 해당하는, 두 테이블을 동일 소유형태 그리고, 동일 가짓수 조건으로 조인하여 각 그룹의 최종 1위를 반환할 수 있도록 하였다. 이러한 패턴을 통해, “per group에서, 최고값 선별”이 가능하도록 하였다. 결과로는, 소유형태,점포 ID,상품 가짓수 세 가지이도록 했고, 동점인 점포가 있으면 모두 표시하도록 했다.

```
7.Variety by Ownership
0.Exit
> 7
```

ownership_type	store_id	kinds
CORPORATE	102	41
FRANCHISE	103	41
CORPORATE	104	41
FRANCHISE	105	41
CORPORATE	106	41
FRANCHISE	107	41
FRANCHISE	109	41
CORPORATE	110	41

```
(base) jiinlee@Jiui-MacBookPro-3 HW_2 % g++ main.cpp \
$(mysql_config --cflags) \
-L/usr/local/mysql/lib -lmysqlclient \
-Wl,-rpath,/usr/local/mysql/lib \
-o main

ld: warning: dylib (/usr/local/mysql/lib/libmysqlclient.dylib) was built for newer macOS vers.
(base) jiinlee@Jiui-MacBookPro-3 HW_2 % ./main

[테이블 레코드 수 요약]
store           : 10
product         : 41
vendor          : 14
customer        : 15
sale            : 60
sale_line_item  : 120
stock           : 408
order_list      : 3
```

사용자는 콘솔 메뉴를 통해 숫자를 입력하고, 추가 인자가 필요할 때는 `std::getline()`으로 받는다. 빈 문자열이나 정해지지 않은 메뉴 번호는 즉시 “잘못 선택” 메시지를 반환하도록 했다. 결과 출력은 `std::setw()`로 열 폭을 조절할 수 있도록 했다. 또한, 애플리케이션이 시작되면 가장 먼저 `verifyMinimums()`가 실행되어 8개 핵심 테이블의 레코드 수를 요약해 보여줄 수 있도록 했다. 그리고 만약, 이 단계에서 0 행이 감지되면 “초기 데이터 로드 오류”를 즉시 인지할 수 있게 했다.

6.5 Testing and Validation (10%)

6.5.1 Test case descriptions and results

테스트는 실제 더미 데이터를 기반으로 데이터베이스가 설계한 제약과 트리거를 올바르게 적용하고, 알맞은 7개의 쿼리 결과를 내놓는지에, 초점을 맞추었다. 먼저, 더미 데이터 INSERT 구문을 통해 14개의 공급사(vendor), 41개의 상품(product), 10개의 점포(store), 15명의 고객(customer), 408건의 재고(stock), 60건의 판매 기록(sale) 및 120건의 영수증 품목(sale_line_item), 그리고 3건의 발주 이력(order_list)을 준비했다.

이러한 환경에서 Test case -1을 실행하여 재고가 충분한 상황(예: stock에서 inventory_level = 40인 상품)에 수량 1을 INSERT하였고, 기대대로 `sk.inventory_level`이 40→39로 감소되며 COMMIT이 정상 완료되는 것을 확인 할 수 있었다. TC-2에서는 동일한 레코드에 quantity = 9999를 INSERT하려 했을 때, `trg_stock_deduct` 트리거가 “재고 부족”을 발생시킴을 확인할 수 있었다.

```
/*-----
(1) 재고 차감
-----*/
DROP TRIGGER IF EXISTS trg_stock_deduct $$
CREATE TRIGGER trg_stock_deduct
BEFORE INSERT ON sale_line_item
FOR EACH ROW
BEGIN
    DECLARE cur_stock INT;

    SELECT sk.inventory_level
    INTO cur_stock
    FROM stock sk
    JOIN sale s ON s.sale_id = NEW.sale_id
    WHERE sk.product_upc = NEW.product_upc
    AND sk.store_id = s.store_id
    FOR UPDATE;

    IF cur_stock IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '해당 점포에 재고 기록이 없습니다.';
    ELSEIF cur_stock < NEW.quantity THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '재고 부족';
    ELSE
        UPDATE stock sk
```

또한, TestCase-3과 Testcase-4로는 sale_line_item의 마지막 품목을 0으로 UPDATE하거나 DELETE 시도할 때, trg_sli_last_zero와 trg_sli_last_del이 각각 “수량 0 불가” 및 “마지막 품목 삭제 불가” 오류를 반환하며 무결성을 보장함을 확인하였다.

그리고 이미 6.4.2 Query Implementation에서, 쿼리 테스트 케이스 실행 결과를 스크린샷 첨부하였지만, 일부만 언급하자면, q3_bestStoreThisQuarter() 쿼리를 실행하여 부산서면점(store_id = 105)이 169,050.00원의 매출을 기록하고, 이게 실제적인 매출과 일치하는 것을 확인했다.이 밖에도, 이처럼 7개 케이스 모두, 실제 데이터와 부합하게 알맞은 결과를 내놓았다.

6.5.2 비즈니스-규칙 검증

비즈니스 규칙은 데이터베이스 레벨에서 자동으로 강제되도록 설계되었으며, 더미 데이터를 통해 실제로 알맞게 작동하는지 확인했다.

먼저, 재고 차감 로직은 trg_stock_deduct 트리거를 통해, 실시간 재고 동기화를 보장할 수 있도록 했고, 재고 기록이 없는 경우에도 “해당 점포에 재고 기록이 없습니다.”라는 메시지로 예외를 처리할 수 있도록 했다. 다음으로, 모든 영수증에는 최소 1개의 line item이 유지되어야 한다는 규칙이 trg_sli_last_zero와 trg_sli_last_del로 UPDATE/DELETE를 수행할 때 강제될 수 있도록 하였다.

외래키 제약에서는 product.vendor_id, sale.store_id, sale_customer.sale_id, order_list(product_upc, store_id) 등에 ON UPDATE CASCADE와 ON DELETE RESTRICT/CASCADE를 적절히 적용하여, 부모 레코드 변경 시 자식 레코드가 자동 갱신 또는 삭제되도록 설정했다.

ENUM과 CHECK 제약은 입력될 수 있는 도메인과 값의 범위를 제한할 수 있도록 하여, 실제로 존재하지 않는 카테고리나, 값의 범위가 데이터로 추가될 수 없도록 하였다.

또한, order_list의 ON DELETE CASCADE 설정은 상품 또는 점포 삭제 시 관련 재고·발주 기록을 자동으로 정리하여 고아 레코드 생성을 방지할 수 있도록 했다.