데이터베이스 시스템

Project 2

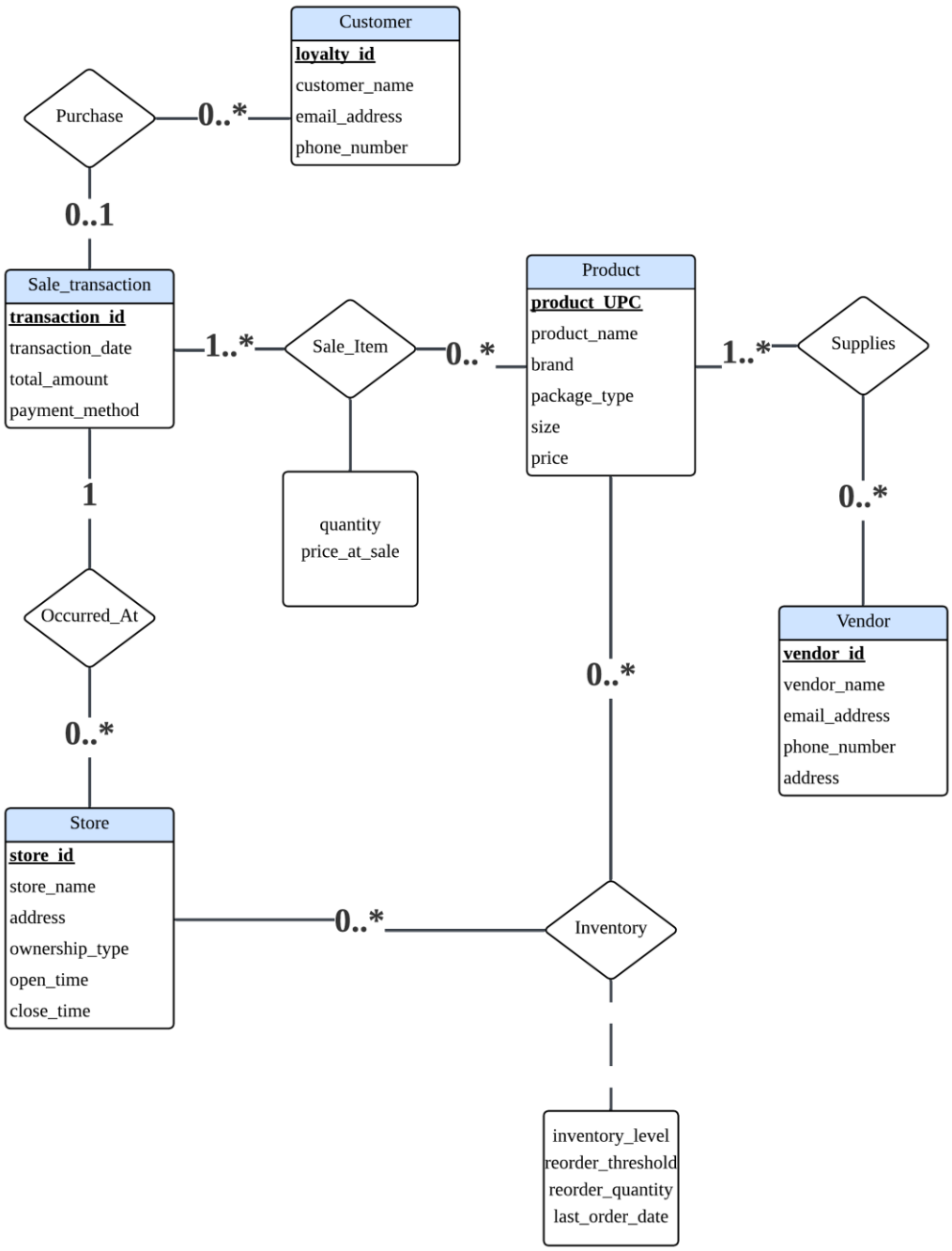
학과 : 컴퓨터공학과

학번 : 20201630

이름 : 장성

1. Logical Schema Design

이 섹션에서는 project1에서 설계한 E-R Diagram을 reduction을 통해서 Logical Shema를 도출하는 과정을 설명합니다.



<E-R Diagram>

E-R Diagram을 Reduction을 이용해서 Logical Schema로 변환하는데 다음과 같은 규칙을 따랐습니다.

Cardinality 관계	변환 방식
1:1	한쪽의 PK를 다른 쪽의 FK로 추가 (보통 total 쪽에)
1:N	N쪽에 1쪽의 PK를 FK로 추가
M:N	관계 테이블 생성, 두 PK를 새로 생성한 테이블의 FK로 추가해서 PK로 사용한다. 이때 Relation의 속성은 테이블에 그대로 추가한다.

우선 Entity는 그대로 테이블로 변환했습니다.

Entity를 테이블로 변환

Customer(loyalty_id (PK), customer_name, email_address, phone_number)

Sale_transaction(transaction_id (PK), transaction_date, total_amount, payment_method)

Product(product_UPC (PK), brand, package_type, size, price)

Vendor(vendor_id (PK), vendor_name, email_address, phone_number, address)

Store(store_id (PK), store_name, address, ownership_type, open_time, close_time)

다음은 realtion을 위 Cardinality 규칙을 따르면서 테이블로 변환했습니다.

Relation을 테이블로 변환

1) Ineventory (M : N 관계)

Store와 Product 엔터티가 M:N 관계이므로 각 엔터티의 PK를 새로 만든 테이블의 FK로 추가하고 추가한 FK의 집합을 PK로 사용합니다. 또한 Inventory의 속성이 있으므로 속성은 그대로 테이블에 추가합니다.

```
Inventory(
    store_id (FK → Store),
    product_UPC (FK → Product),
    inventory_level,
    reorder_threshold,
    reorder_quantity,
    last_order_date,

    PRIMARY KEY(store_id, product_UPC)
)
```

2) Supplies (M : N 관계)

Product와 Vendor 엔터티가 M:N 관계이므로 각 엔터티의 PK를 새로 만든 테이블의 FK로 추가하고 추가한 FK의 집합을 PK로 사용합니다.

```
Supplies(
    product_UPC (FK → Product),
    vendor_id (FK → Vendor),

    PRIMARY KEY(product_UPC, vendor_id)
)
```

3) Occurred_At (1 : N 관계)

1 : N 관계이므로 1의 PK를 N쪽에 FK로 추가한다. 여기서는 Store의 PK인 store_id를 Sale_transaction에 추가합니다.

-변경된 Sale_transaction 테이블

Sale_transaction(

transaction_id (PK),

transaction_date,

total_amount,

payment_method,

store_id (FK → Store)

)

4) Purchase (1: N 관계)

0..1 : N 관계이므로 0..1 쪽의 PK를 N쪽에 nullable FK로 추가합니다.

-변경된 Sale_transaction 테이블

Sale_transaction(

transaction_id (PK),

transaction_date,

total_amount,

payment_method,

store_id (FK → Store),

loyalty_id(FK → Customer)

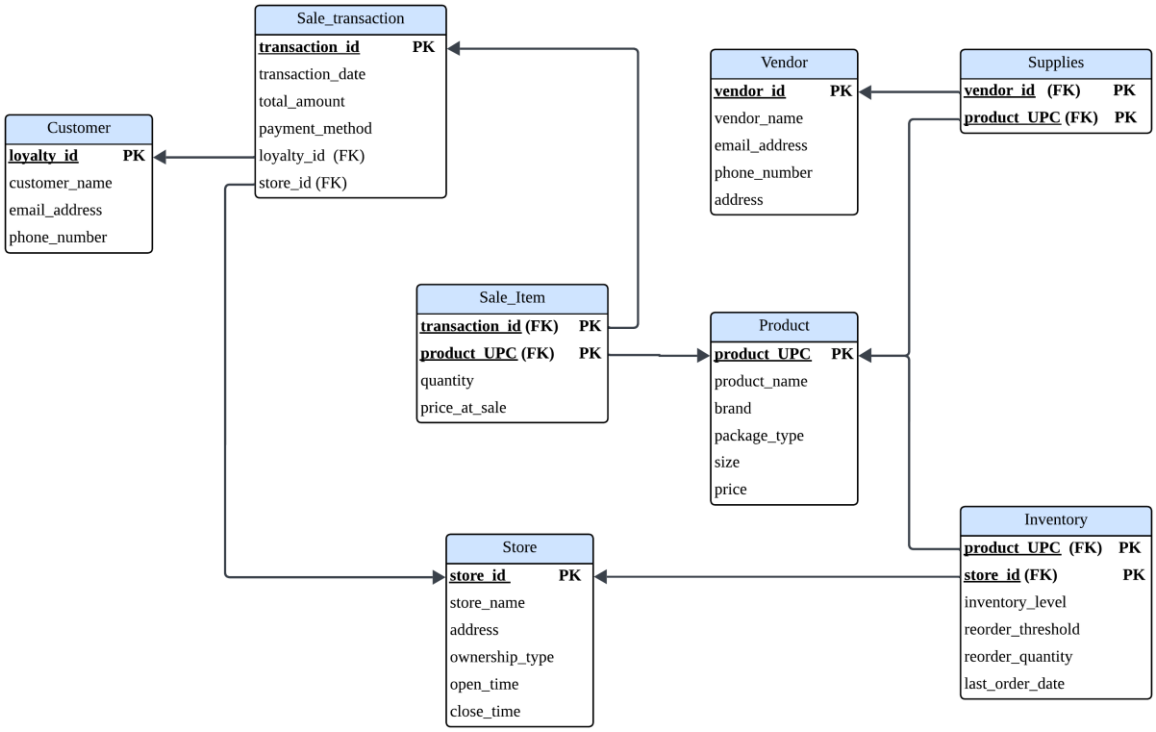
)

5) Sale_Item (M : N 관계)

Sale과 Product 엔터티가 M:N 관계이므로 각 엔터티의 PK를 새로 만든 테이블의 FK로 추가하고 추가한 FK의 집합을 PK로 사용한다. 또한 Sale_Item의 속성이 있으므로 속성은 그대로 테이블에 추가합니다.

```
Sale_Item(
        transaction_id (FK → Sale_Transaction),
        product_UPC FK → Product,
        quantity,
        price_at_sale,
        PRIMARY KEY(transaction_id, product_UPC)
)
```

위 같은 과정을 거쳐서 최종 logical_schema는 다음과 같이 산출됩니다.

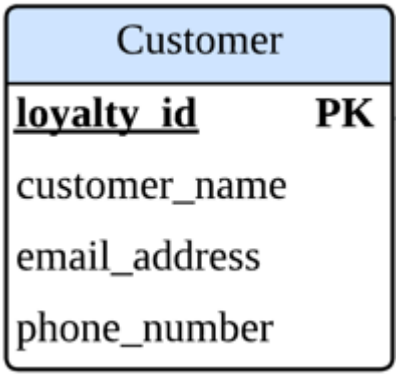


<Logical Schema>

2. Normalization Analysis

다음은 BCNF를 이용해서 작성한 Logical schema를 정규화 하는 과정이다.
우선 각 Relation의 FD를 구하고 이후 BCNF decompostion을 진행해서 정규화를 진행 했습니다.

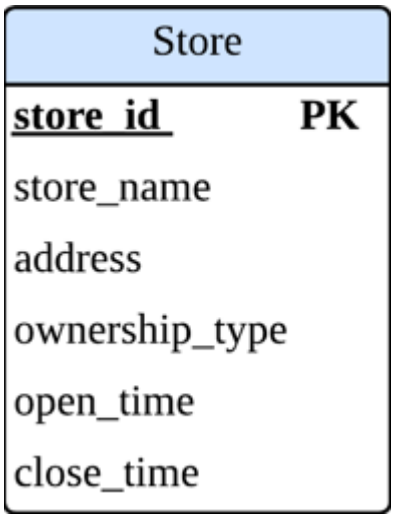
1) Customer relation



FD : loyalty_id → customer_name, email_address, phone_number

loyalty_id 가 PK이므로 BCNF 만족한다.

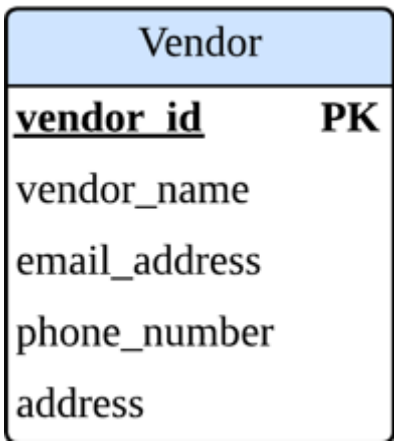
2) Store relation



FD : store_id → store_name, address, ownership_type, open_time, close_time

store_id가 PK이므로 BCNF 만족한다

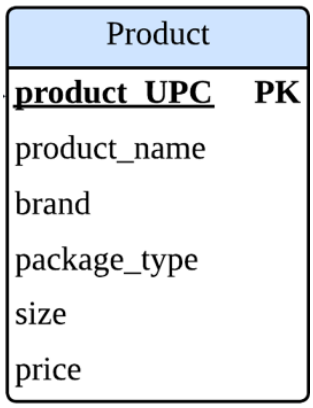
3) Vendor relation



FD : vendor_id → vendor_name, email_address, phone_number, address

vendor_id 가 PK이므로 BCNF 만족한다.

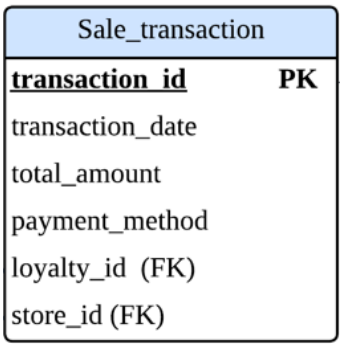
4) Product relation



FD : product_UPC → product_name, brand, package_type, size, price

product_UPC 가 PK이므로 BCNF 만족한다.

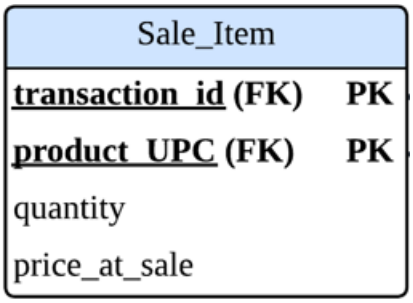
5) Sale_transaction relation



FD : transaction_id → transaction_date, total_amount, payment_method, loyalty_id, store_id

transcation_id 가 PK이므로 BCNF 만족한다.

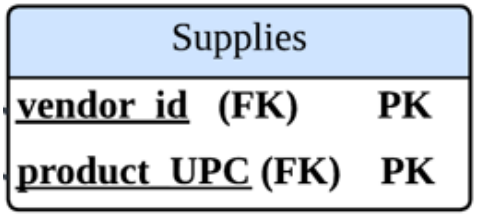
6) Sale_Item relation



FD : (transaction_id, product_UPC) → quantity, price_at_sale

(transaction_id, product_UPC) 가 PK이므로 BCNF 만족한다.

7) Supplies relation



FD : 존재하지 않는다.
FD가 존재하지 않으므로 BCNF를 판단할 FD가 존재하지 않는다. 따라서 BCNF 만족한다.

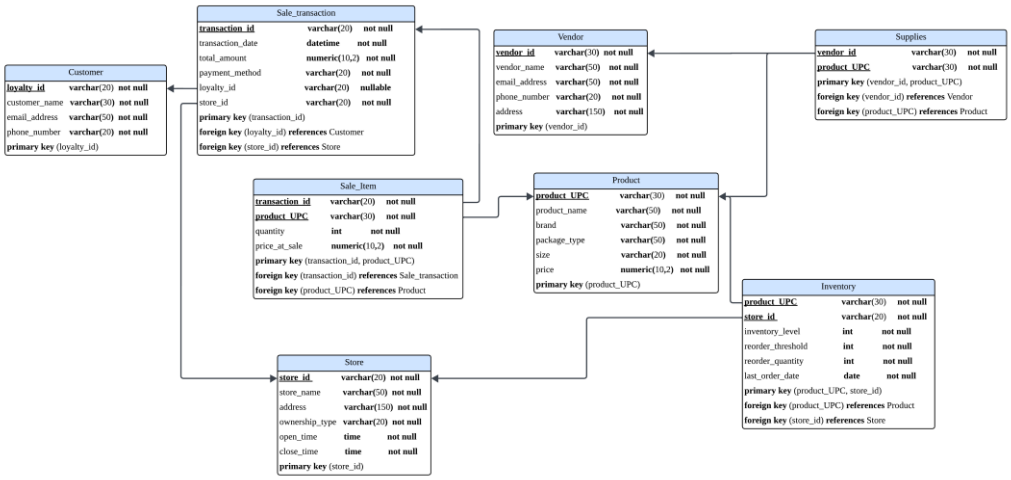
8) Inventory relation

FD : (product_UPC, store_id) → inventory_level, reorder_threshold, reorder_quantity, last_order_date

(product_UPC, store_id) 가 PK이므로 BCNF 만족한다.

따라서 모든 relation 이 BCNF 만족하므로 추가적인 decompostion은 불필요하다.

3. Physical Implementation



<physical schema>

이 섹션에서는 앞서 설계된 논리적 스키마를 MySQL 환경에 최적화된 물리적 스키마로 변환하는 과정과 그 근거에 대해 설명합니다. 또한 데이터베이스 제약 조건을 통해 정의된 비즈니스 규칙이 어떻게 적용되었는지 설명하고 데이터베이스 기능 검증에 사용된 샘플 데이터에 대해 설명합니다.

1) 데이터 유형 선택 근거

효율적인 데이터 저장과 비즈니스 로직의 정확한 구현을 위해 엔티티 내의 각 속성에 대해 적절한 데이터 유형을 선택했습니다. 다음은 주요 데이터 유형 선택의 근거입니다:

- **VARCHAR**: ID, 이름, 주소, 이메일 주소, 전화번호, 브랜드, 패키지 유형 등 가변 길이의 문자열 데이터에 사용한다. 예를 들어 길이가 다양할 수 있는 **loyalty_id, store_id, product_UPC, vendor_id, transaction_id** 와 같은 식별자나 **customer_name, store_name, product_name, vendor_name, address** 같은 일반 텍스트 정보에 적합하여 적용했습니다.

- **INT**: **quantity, inventory_level, reorder_threshold, reorder_quantity** 등 정수 값을 저장하는 데 사용합니다. 이러한 값은 음수가 될 수 없으므로 >= 0 또는 > 0을 강제하는 **CHECK 제약** 조건을 추가하여 데이터 무결성을 강화했다.

- **NUMERIC(10,2**): 소수점 이하 두 자리까지 정확도가 중요한 **price, total_amount, price_at_sale** 과 같은 화폐 값에 사용했습니다. 가격과 금액이 음수가 아닌지 확인하기 위해 **CHECK(column_name >= 0)** 제약 조건이 포함되었습니다.

- **TIME**: 시간: 매장 운영 시간(**open_time, close_time**)과 같이 시간 정보만 필요한 속성에 사용합니다.

- **DATE**: **last_order_date**와 같이 날짜 정보만 필요한 속성에 사용됩니다.

- **DATETIME**: 트랜잭션이 발생한 정확한 순간을 기록하는 데 필수적인 날짜와 시간 정보를 모두 저장하기 위해 **transaction_date**에 사용합니다.

2) 제약 조건 구현 및 비즈니스 규칙 적용

설계된 논리적 스키마의 비즈니스 규칙들은 물리적 스키마 생성 시 다양한 제약 조건(Constraints)으로 구현되어 데이터의 무결성을 보장합니다.

- **기본 키 (Primary Key)**: 각 테이블의 고유 식별자 역할을 하는 컬럼(예: Customer의 loyalty_id, Store의 store_id, Product의 product_UPC 등)은 PRIMARY KEY로 지정했습니다. 복합 기본 키는 Sale_Item (transaction_id, product_UPC), Supplies (vendor_id, product_UPC), Inventory (product_UPC, store_id) 테이블에 적용되어 다대다 관계를 효과적으로 표현하고 각 튜플의 고유성을 보장합니다.

- **외래 키 (Foreign Key)**: 테이블 간의 관계를 정의하고 참조 무결성을 유지하기 위해 FOREIGN KEY 제약 조건이 사용되었습니다. 예를 들어, Sale_transaction 테이블의 loyalty_id는 Customer 테이블의 loyalty_id를, store_id는 Store 테이블의 store_id를 참조하도록 설정되었습니다. 이는 유효하지 않은 고객 또는 매장 ID로의 거래 기록을 방지합니다.

- **NOT NULL**: 필수적으로 값이 존재해야 하는 컬럼(예: 모든 기본 키 컬럼, 이름, 주소, 필수적인 시간 및 날짜 등)에는 NOT NULL 제약 조건을 부여하여 데이터 누락을 방지했습니다. 그리고 NULL이 들어가면 성능이 떨어지기 때문에 웬만한 데이터에 모두 적용 했습니다.

- **NULLABLE :** Sales_transaction에서 loyalty_id 속성에 사용합니다. 비회원 구매를 고려한 선택이었습니다.

- **CHECK**: 특정 컬럼의 값이 미리 정의된 규칙을 따르도록 CHECK 제약 조건을 활용하여 비즈니스 규칙을 강제했습니다.
    - Customer, Vendor 테이블의 email_address: email_address like '%@%' 조건을 통해 유효한 이메일 형식(@ 포함)을 강제합니다.
    - Product, Sale_transaction, Sale_Item 테이블의 금액 관련 컬럼(price, total_amount, price_at_sale): column_name >= 0 조건을 통해 가격 및 금액이 음수가 될 수 없도록 합니다.
    - Sale_Item 테이블의 quantity: quantity > 0 조건을 통해 판매 수량이 0보다 커야 함을 강제합니다.
    - Store 테이블의 ownership_type: ownership_type in ('Franchise', 'Corporate') 조건을 통해 소유권 유형이 'Franchise' 또는 'Corporate' 중 하나만 허용되도록 제한합니다.
    - Sale_transaction 테이블의 payment_method: payment_method in ('cash', 'card', 'online') 조건을 통해 결제 방식이 지정된 세 가지 중 하나만 허용되도록 제한합니다.
    - Inventory 테이블의 재고 관련 컬럼(inventory_level, reorder_threshold, reorder_quantity): column_name >= 0 조건을 통해 해당 값들이 음수가 될 수 없음을 강제합니다.

이러한 제약 조건들은 데이터베이스에 저장되는 데이터의 정확성과 일관성을 보장하며, 비즈니스 규칙이 시스템 수준에서 자동으로 준수되도록 합니다.

3) 샘플 데이터 설명

3-1) **Customer** (loyalty_id, customer_name, email_address, phone_number)

('CUST001', 'Kim Jihye', 'jihye.kim@example.com', '010-1234-5678')

회원 번호인 loyalty_id는 " CUST + 번호 " 로 구성했다.

3-2) **Store** (store_id, store_name, address, ownership_type, open_time, close_time)

('STR001', 'CU Gangnam', '123 Teheran-ro, Seoul', 'Corporate', '00:00:00', '23:59:59')

매장의 고유 아이디인 store_id는 "STR + 번호" 로 구성했으며 매장 이름인 store_name은 실제 존재하는 "편의점 브랜드명 + 지역명"으로 각 편의점의 지점명으로 표현했다.
이에 맞춰서 편의점의 주소를 실제 한국 주소로 나타내었다.

3-3) Vendor (vendor_id, vendor_name, email_address, phone_number, address)

('VEND001', 'Orion Distributors', 'orion_dist@example.com', '02-1111-2222', 'Seoul, Guro-gu, Digital-ro 26-gil 98'),

"VEND + 숫자"로 고유한 vedor_id를 구성했다. 또한 실제 한국 주소로 address 데이터를 삽입했다.

3-4) Product (product_UPC, product_name, brand, package_type, size, price)

('8801234567890', 'Choco Pie', 'Orion', 'Box', '12pcs', 4.50)

product_UPC는 바코드와 같이 고유한 숫자들의 조합으로 표현했으며 브랜드(brand)와 상품명(product_name), 포장 방식(package_type), 용량(size)는 실제 유통되고 있는 상품들을 참고하여 작성했다.

3-5) Supplies (vendor_id, product_UPC)

('VEND001', '8801234567890')

외래키의 참조제약에 알맞게 vedor와 product 테이블에 사용한 값들을 넣었다.

3-6) Inventory (product_UPC, store_id, inventory_level, reorder_threshold, reorder_quantity, last_order_date)

('8801234567890', 'STR001', 15, 10, 20, '2025-05-20')

외래키의 참조제약에 알맞게 product, store 테이블에 삽입했던 데이터에 알맞게 데이터를 삽입했다.

3-7) Sale_transaction (transaction_id, store_id, loyalty_id, transaction_date, total_amount, payment_method)

('TRN00001', 'STR001', 'CUST001', '2025-05-15 10:00:00', 10.50, 'card')

외래키의 참조제약에 알맞게 customer, store 테이블에 삽입했던 데이터에 알맞게 데이터를 삽입했다. transaction_id는 "TRN+숫자"로 구성했다. 또한 payment_method는 card, cash 그리고 online 이렇게 3가지 방법을 사용하는 걸로 가정했다.

3-8) Sale_Item (transaction_id, product_UPC, quantity, price_at_sale)

('TRN00001', '8801234567890', 2, 9.00)

외래키의 참조제약에 알맞게 sale_transaction, product 테이블에 삽입했던 데이터에 알맞게 데이터를 삽입했다.

price_at_sale은 할인율 등을 고려하여 상품의 정가인 price와 다르게 넣었다.

```
create table Customer (
    loyalty_id      varchar(20)     not null,
    customer_name   varchar(30)     not null,
    email_address   varchar(50)     not null check (email_address like '%@%'),
    phone_number    varchar(20)     not null,
    primary key (loyalty_id)
);
create table Store (
    store_id        varchar(20)     not null,
    store_name      varchar(50)     not null,
    address         varchar(150)    not null,
    ownership_type  varchar(20)     not null check (ownership_type in ('Franchise', 'Corporate')),
    open_time       time            not null,
    close_time      time            not null,
    primary key (store_id)
);

create table Vendor (
    vendor_id       varchar(30)     not null,
    vendor_name     varchar(50)     not null,
    email_address   varchar(50)     not null check (email_address like '%@%'),
    phone_number    varchar(20)     not null,
    address         varchar(150)    not null,
    primary key (vendor_id)
);
```

```sql
create table Product (
    product_UPC     varchar(30)     not null,
    product_name    varchar(50)     not null,
    brand           varchar(50)     not null,
    package_type    varchar(50)     not null,
    size            varchar(20)     not null,
    price           numeric(10,2)   not null check (price >= 0),
    primary key (product_UPC)
);

create table Sale_transaction (
    transaction_id    varchar(20)     not null,
    transaction_date  datetime        not null,
    total_amount      numeric(10,2)   not null check (total_amount >= 0),
    payment_method    varchar(20)     not null check (payment_method in ('cash', 'card', 'online')),
    loyalty_id        varchar(20),
    store_id          varchar(20)     not null,
    primary key (transaction_id),
    foreign key (loyalty_id) references Customer(loyalty_id),
    foreign key (store_id) references Store(store_id)
);
create table Sale_Item (
    transaction_id  varchar(20)     not null,
    product_UPC     varchar(30)     not null,
    quantity        int             not null check (quantity > 0),
    price_at_sale   numeric(10,2)   not null check (price_at_sale >= 0),
    primary key (transaction_id, product_UPC),
    foreign key (transaction_id) references Sale_transaction(transaction_id),
    foreign key (product_UPC) references Product(product_UPC)
);

create table Supplies (
    vendor_id       varchar(30)     not null,
    product_UPC     varchar(30)     not null,
    primary key (vendor_id, product_UPC),
    foreign key (vendor_id) references Vendor(vendor_id),
    foreign key (product_UPC) references Product(product_UPC)
);
```

```sql
create table Inventory (
    product_UPC         varchar(30)     not null,
    store_id            varchar(20)     not null,
    inventory_level     int             not null check (inventory_level >= 0),
    reorder_threshold   int             not null check (reorder_threshold >= 0),
    reorder_quantity    int             not null check (reorder_quantity >= 0),
    last_order_date     date            not null,
    primary key (product_UPC, store_id),
    foreign key (product_UPC) references Product(product_UPC),
    foreign key (store_id) references Store(store_id)
);
```

4. Application Development

이 섹션에서는 main.cpp에서 MySQL 데이터베이스에 연결한 방법과 쿼리 구현 방식, 그리고 인터페이스에 대해 설명합니다.

1. 데이터베이스 연결

MySQL C API를 사용하여 데이터베이스 연결을 구현했습니다.

라이브러리 및 헤더 파일 포함: MySQL C API를 사용하기 위해 mysql.h 헤더 파일을 포함합니다. 컴파일 시에는 MySQL 클라이언트 라이브러리(libmysql.lib)와 연결됩니다. 이를 위해 빌드 명령어에 -I (include path)와 -L (library path) 옵션을 사용하여 MySQL 개발 라이브러리의 위치를 명시했습니다.

  - **헤더 파일**: #include <mysql.h>
  - **컴파일 및 링크**: g++ main.cpp -o main -I"C:/Program Files/MySQL/MySQL Server 8.0/include" -L"C:/Program Files/MySQL/MySQL Server 8.0/lib" -lmysql

- **연결 객체 초기화**: MYSQL *conn; 변수를 선언하고 mysql_init(NULL) 함수를 호출하여 MySQL 연결 객체를 초기화합니다. 이 핸들은 이후 모든 데이터베이스 작업에 사용됩니다. 마치 c의 파일포인터와 같이 사용됩니다. (FILE* fp)
- **데이터베이스 연결**: mysql_real_connect() 함수를 사용하여 실제 데이터베이스 연결을 설정합니다. 이 함수에는 MySQL 서버의 호스트 주소(localhost), 사용자 이름, 비밀번호, 데이터베이스 이름, 포트 번호, 유닉스 소켓 경로, 클라이언트 플래그 등의 인

자가 전달됩니다. 연결 성공 여부는 반환 값이 NULL인지 아닌지로 판단합니다. 반환 값이 NULL인 경우 에러를 출력하고 mysql_error(conn) 함수를 이용해서 에러 메시지 출력 후, 데이터베이스와의 연결을 끊고 프로세스를 종료합니다.

    - mysql_real_connect(conn, "localhost", "user", "password", "database_name", 3306, NULL, 0);

- **연결 종료 및 리소스 해제**: 유저에게 0을 입력받으면 mysql_close(conn) 함수를 호출하여 열린 연결을 닫습니다. 또한 mysql_free_result(res)함수를 이용해서 저장되어 있는 결과값이 차지하고 있는 메모리를 해제해줍니다.

2. 쿼리 구현

사용자의 선택(TYPE1 ~TYPE7)에 따라 다양한 SQL 쿼리를 실행할 수 있도록 구현되었습니다.

- **쿼리 준비**: void init_query(string *query) 함수를 사용하여 미리 SQL 쿼리 기본틀을 만들어 query배열에 저장합니다. TYPE1과 TYPE6과 같이 사용자 입력이 필요한 쿼리의 경우, 사용자 입력을 받아 쿼리를 완성합니다. 나머지 TYPE은 미리 완성된 쿼리 문자열을 사용합니다.
- **쿼리 실행**:
    - mysql_query(conn, sql_query.c_str()) 함수를 사용하여 준비된 SQL 쿼리를 데이터베이스 서버로 전송하고 실행합니다. 실패할 경우 mysql_error(conn) 함수를 이용해서 에러 메시지를 출력하고 프로세스를 종료합니다.
    - sql_query.c_str() 함수는 C++ std::string 객체를 C 스타일 문자열(const char*)로 변환하여 mysql_query 함수가 요구하는 형식에 맞춥니다.
- **쿼리 결과 처리**:
    - mysql_store_result(conn) 함수를 호출하여 SELECT 쿼리 실행 결과를 클라이언트로 가져와 MYSQL_RES 타입의 res 변수에 저장합니다. 이 함수는 결과가 없거나 오류가 발생하면 NULL을 반환합니다.
    - mysql_num_fields(res)를 사용하여 결과 집합의 컬럼 수를 가져옵니다.

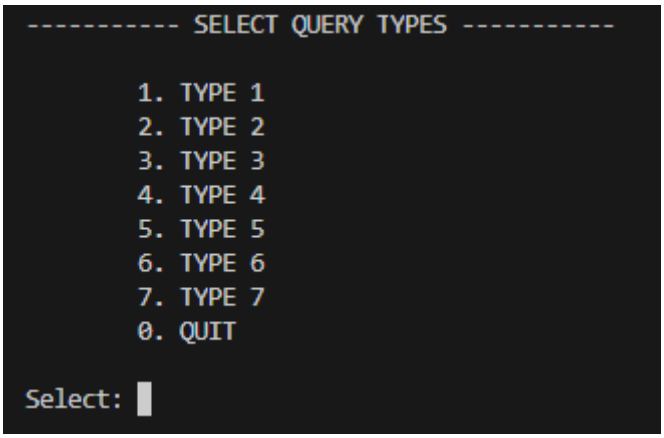
      사용 예시: int num_fields = mysql_num_fields(res);

- mysql_fetch_fields(res)를 사용하여 각 컬럼의 메타데이터(이름 등)를 가져와 헤더를 출력합니다.

  사용 예시: MYSQL_FIELD *fields = mysql_fetch_fields(res);

- mysql_fetch_row(res)를 반복적으로 호출하여 결과 집합의 각 행을 탐색합니다. 각 행은 MYSQL_ROW 타입으로 반환되며, 배열 형태로 특정 속성 값에 접근할 수 있습니다.
- cout << left << setw(40)과 같은 C++ 스트림 조작을 사용하여 결과 테이블의 컬럼을 왼쪽 정렬하고 고정된 너비(40)로 출력하여 가독성을 높입니다.

5. 실행 결과



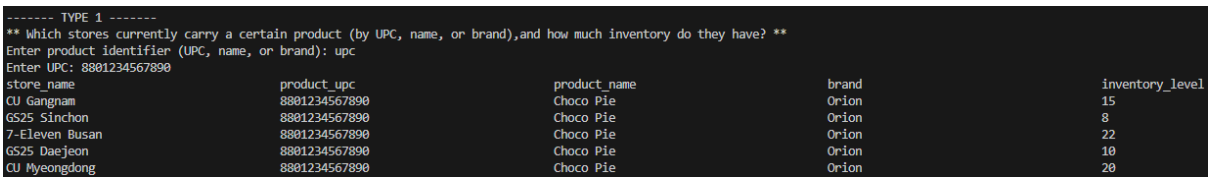
기본 유저 입력 창

1) TYPE 1

UPC, product name, brand를 입력하면 입력한 항목에 해당하는 물품을 갖고있는 모든 가게의 해당 상품의 재고가 표시됩니다.

- UPC 입력

UPC 8801234567890 (Choco Pie)을 보유하고 있는 모든 매장의 해당 삼품의 재고 정보가 표시됩니다.



-name 입력

Choco Pie 입력: Choco Pie(UPC 8801234567890)를 보유하고 있는 모든 매장의 해당 상품의 재고 정보가 표시됩니다.

```
------- 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): brand
Enter brand: Orion
store_name                product_upc          product_name          brand          inventory_level
CU Gangnam                8801234567890        Choco Pie             Orion          15
GS25 Sinchon              8801234567890        Choco Pie             Orion          8
7-Eleven Busan            8801234567890        Choco Pie             Orion          22
GS25 Daejeon              8801234567890        Choco Pie             Orion          10
CU Myeongdong             8801234567890        Choco Pie             Orion          20
```

- brand 입력

  Orion 입력: 브랜드명이 orion인 상품을 보유하고 있는 모든 매장의 해당 상품의 재고 정보가 표시됩니다. 현재 데이터베이스에는 브랜드명이 Orion인 제품은 Choco Pie밖에 없다는 것도 알 수 있습니다.

```
------- 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): brand
Enter brand: Orion
store_name                product_upc          product_name          brand          inventory_level
CU Gangnam                8801234567890        Choco Pie             Orion          15
GS25 Sinchon              8801234567890        Choco Pie             Orion          8
7-Eleven Busan            8801234567890        Choco Pie             Orion          22
GS25 Daejeon              8801234567890        Choco Pie             Orion          10
CU Myeongdong             8801234567890        Choco Pie             Orion          20
```

2) TYPE 2

   지난 1달 동안 각 점포에서 가장 많이 팔린 물품을 보여줍니다. STR011 부터 STR021은 거래가 없었기 때문에 결과에 포함되지 않았습니다.

```
------- TYPE 2 -------
** Which products have the highest sales volume in each store over the past month? **
store_id                  store_name           product_UPC           product_name          total_sales_volume
STR003                    7-Eleven Busan       8807777888899         Ramen                 5
STR008                    7-Eleven Jeju        8803030303030         Candy C               5
STR001                    CU Gangnam           8801234567890         Choco Pie             11
STR004                    CU Hongdae           8801010101010         Snack A               10
STR006                    CU Myeongdong        8801010101010         Snack A               3
STR009                    CU Yeouido           8801234567890         Choco Pie             2
STR005                    GS25 Daejeon         8801111222233         Milk                  5
STR010                    GS25 Incheon         8804040404040         Water                 2
STR007                    GS25 Jongno          8809876543210         Coca-Cola             4
STR002                    GS25 Sinchon         8809876543210         Coca-Cola             8
```

3) TYPE 3

   가장 높은 수익을 낸 지점을 출력합니다.

```
------- TYPE 3 -------
** Which store has generated the highest overall revenue this quarter? **
store_id                  store_name           total_revenue
STR001                    CU Gangnam           78.90
```

4) TYPE 4

가장 많은 물품을 공급하는 업체와 그 업체가 공급한 물체가 총 얼마만큼 팔렸는지 출력합니다.

```
------- TYPE 4 -------
** Which vendor supplies the most products across the chain, and how many total units have been sold? **
vendor_id                       vendor_name                     num_supplied_products           total_sold_quantity
VEND004                         Food & Beverage Inc.            16                              93
```

5) TYPE 5

각 점포가 갖고있는 재고가 재주문 임계치보다 낮은 경우, 해당 점포와 물품, 재고 현황, 재주문 임계치 , 재주문 수량 등을 출력해줍니다.

```
------- TYPE 5 -------
** Which products in each store are below the reorder threshold and need restocking? **
store_id                        store_name                      product_UPC                     product_name                    inventory_level                 reorder_threshold               reorder_quantity
STR001                          CU Gangnam                      8804040404040                   Water                           5                               10                              10
STR001                          CU Gangnam                      8805555666677                   Coffee                          8                               10                              20
STR001                          CU Gangnam                      8800876543210                   Coca-Cola                       3                               5                               30
STR002                          GS25 Sinchon                    8801111222233                   Milk                            4                               5                               50
STR002                          GS25 Sinchon                    8801234567890                   Choco Pie                       8                               10                              15
STR003                          7-Eleven Busan                  8801111222233                   Milk                            0                               5                               50
STR003                          7-Eleven Busan                  8807531086429                   Soap Bar                        2                               5                               10
STR005                          GS25 Daejeon                    8807531086429                   Soap Bar                        3                               5                               5
STR005                          GS25 Daejeon                    8880909090890                   Bread E                         1                               5                               15
STR010                          7-Eleven Sokcho                 8809753100642                   Energy Drink                    3                               5                               10
STR020                          CU Sejong                       8807531086429                   Soap Bar                        1                               5                               10
```

6) TYPE 6

회원 구매내역 중에 커피와 주로 어떤 제품을 같이 구매하는 지 출력해줍니다.
커피의 상품명을 입력하면 결과를 출력해줍니다. 현재 데이터에는 커피가 Coffee밖에 없으므로 Coffee를 입력했습니다.

```
------- TYPE 6 -------
** List the top 3 items that loyalty program customers typically purchase with coffee. **
Enter a coffee product name: Coffee
product_UPC                     product_name                    co_purchase_count
8801234567890                   Choco Pie                       2
8803030303030                   Candy C                         1
8801010101010                   Snack A                         1
```

7) TYPE 7

본사 직영점과 가맹점 중, 각각 가장 많은 종류의 상품을 취급하고 있는 매장을 출력해줍니다.

```
------- TYPE 7 -------
** Among franchise-owned stores, which one offers the widestvariety of products, and how does that compare to corporate-owned stores? **
ownership_type                  store_id                        store_name                      distinct_product_count
Franchise                       STR002                          GS25 Sinchon                    8
Corporate                       STR001                          CU Gangnam                      8
```

8) QUIT

프로그램을 종료합니다.

```
----------- 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: 0
Program terminating
```