

# DB\_project2\_report

[목차]

## 1. Logical Schema Design

1-1 최종 Entity

1-2. 변동 사항

1-3. Relationship

1-3. Entity 별 설명

## 2. Normalization Analysis

## 3. Physical Implementation

## 4. Application Development

4-1. Database Connectivity Implementation Details

4-2. Query Implementation

4-3. Error Handling and User Interface Design

## 5. Testing and Validation

5-1 Test Case Description and Results

## 6. 결론

# 1. Logical Schema Design

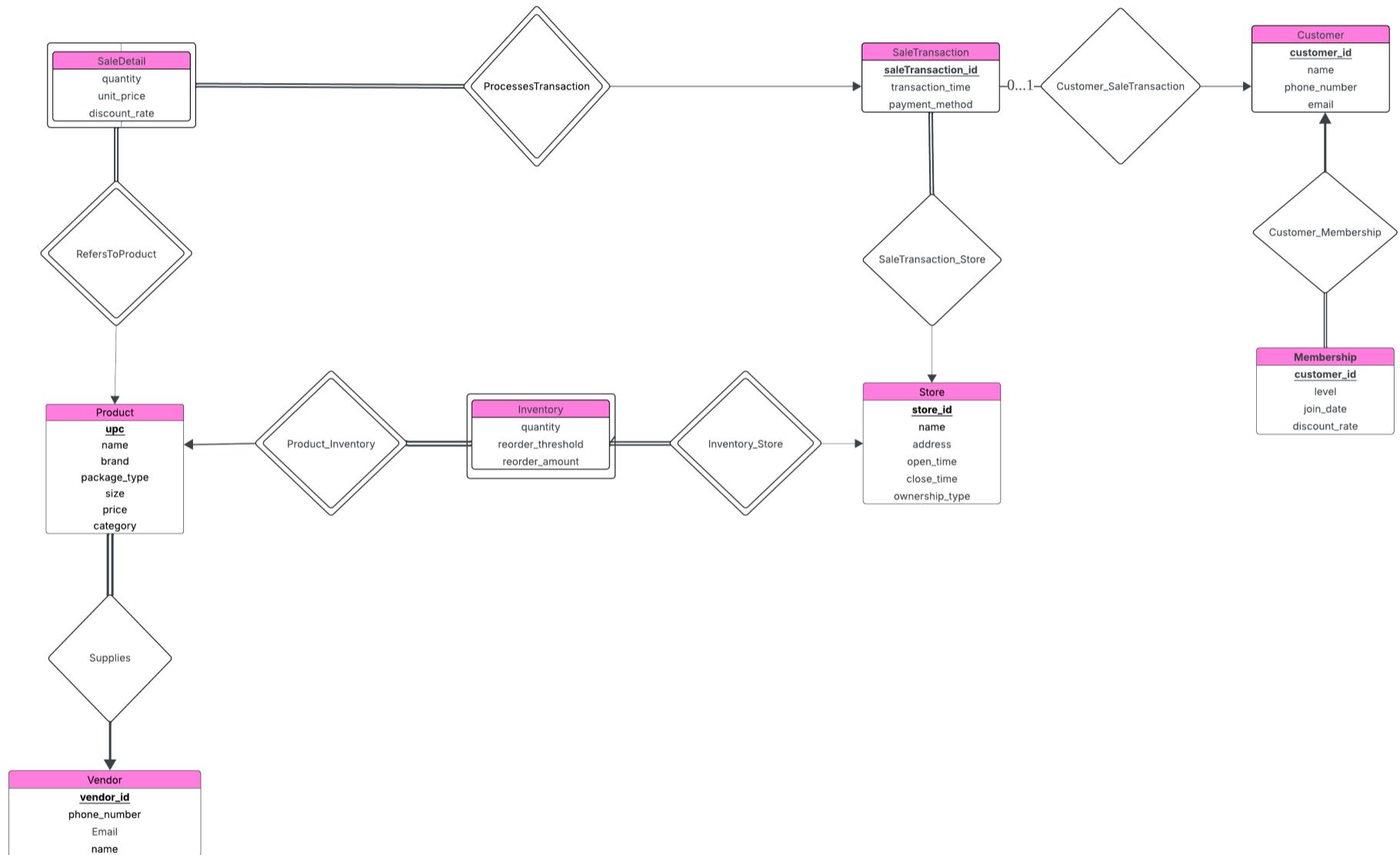


그림 1. logical schema

## 1-1 최종 Entity

Entity	Primary Key	Attributes	Foreign Key
Store	store_id	name, address, open_time, close_time, ownership_type	-
Product	upc	name, brand, package_type, size, price, category	vendor_id
Vendor	vendor_id	name, phone_number, Email	-
Inventory	(store_id, upc)	quantity, reorder_threshold, reorder_amount	store_id, upc

Entity	Primary Key	Attributes	Foreign Key
Customer	customer_id	name, <b>phone_number</b> , email	-
Membership	<b>customer_id</b>	level, join_date, discount_rate, customer_id	customer_id
SaleTransaction	saleTransaction_id	transaction_time, payment_method	store_id, customer_id (optional)
SaleDetail	(saleTransaction_id, upc)	quantity, unit_price, discount_rate	saleTransaction_id, upc

## 1-2. 변동 사항

변경 대상	변경 전	변경 후	변경 이유
Membership 테이블	<b>membership_id</b> 존재	<b>membership_id</b> 삭제	customer_id를 PK로 활용 (추후 보충 설명)
Customer 테이블	<b>phone</b>	<b>phone_number</b>	변수명 통일 및 일관성 유지 (다른 테이블과 통일)

## 1-3. Relationship

- Relationship 요약 Table

관계 이름	연결된 엔티티	Cardinality	Participation	Weak Entity
Inventory_Store	Inventory – Store	N:1	Total (Inventory 측)	✗
Product_Inventory	Inventory – Product	N:1	Total (Inventory 측)	✗
Supplies	Vendor – Product	1:N	Partial (Product 측)	✗
Customer_Membership	Customer – Membership	1:1	Total (Membership 측), Partial (Customer 측)	<input checked="" type="checkbox"/> (Membership)
Customer_SaleTransaction	Customer – SaleTransaction	1:N (optional)	Partial (SaleTransaction 측)	✗
SaleTransaction_Store	SaleTransaction – Store	N:1	Total (SaleTransaction 측)	✗
ProcessesTransaction	SaleTransaction – SaleDetail	1:N	Total (SaleDetail 측)	<input checked="" type="checkbox"/> (SaleDetail)
RefersToProduct	SaleDetail – Product	N:1	Total (SaleDetail 측)	✗

## 1-3. Entity 별 설명

### • Store

각 매장은 고유한

**store\_id**로 식별됩니다. 이름, 주소, 운영 시간(open\_time, close\_time) 등의 속성을 통해 위치 및 영업 정보를 관리할 수 있습니다.

또한 Store는 Inventory, SaleTransaction 등 핵심 트랜잭션과의 연결점이 되어, 매장별 재고 관리, 판매 이력 집계, 고객 유입 분석 등에 활용됩니다.

#### ◦ 주요 Attribute 설명

**ownership\_type** : 속성을 통해 프랜차이즈와 직영 여부를 구분하기 위한 필드로, 매장 운영 형태에 따른 정책 차별화나 성과 비교를 할 수 있는 근거 데이터로 활용될 수 있습니다. 매장은 제품과의 관계에서 재고를 관리하며, 판매 기록의 출발점 역할을 합니다.

**open\_time** & **close\_time** : 24시 운영 매장은 open\_time과 close\_time을 00:00으로 설정하여 구분할 수 있습니다.

### • Product

편의점에서 판매되는 모든 개별 품목을 추상화한 엔티티입니다. upc, 제품명, 브랜드, 패키지 타입, 용량(사이즈), 가격 등 다양한 속성을 포함합니다.

각 제품은 여러 매장에서 판매될 수 있으며, 동일한 제품이라 하더라도 매장에 따라 가격, 진열 여부, 재고 수량이 달라질 수 있습니다. 오직 한 공급업체에 의해 공급되도록 가정하였습니다.

#### ◦ 주요 attribute 설명

**upc** :(Universal Product Code), 각 제품의 고유 식별자

**category** : 식품, 음료, 비식품 등으로 구분하기 위한 속성으로, 카테고리별 매출, 재고 회전율 등에 필요하다고 판단하여 추가하였습니다.

**vendor\_id** : 외래키로 보유하여, 해당 제품을 공급하는 업체와 직접적으로 연결되며, 공급업체별 납품 제품 수, 품목별 공급 안정성 등의 분석이 가능합니다.

- **Vendor**

`vendor_id` 를 기본 키로 가지며, 업체명, 연락처 등의 속성을 포함합니다. 제품 공급업체로, 하나의 Vendor는 여러 개의 제품을 공급할 수 있는 1:N 관계 구조입니다.

- **Inventory**

Store와 Product 사이의 다대다 관계를 분해하기 위해 도입된

중간 엔티티입니다. 하나의 매장에 어떤 제품이 몇 개 남아 있는지를 추적하며, `quantity`, `reorder_threshold`, `reorder_amount` 등의 속성을 통해 재고 부족 알림, 자동 발주 시스템을 구현할 수 있습니다. 이 엔티티는 제품별 재고 수량, 입고 상태, 재고 회전율 등을 분석하는 데 필요한 데이터를 제공합니다

- 주요 attribute 설명

`reorder_threshold` & `reorder_amount`: `quantity`가 해당 `reorder_threshold` 낮을 경우 `reorder_amount` 만큼 자동 재 주문을 하도록 설정할 수 있습니다.

- **Customer**

멤버십 및 구매 분석을 위해 설계된 엔티티입니다. 고객 ID로 식별하며, 추후 마케팅에 활용하기 위한 이름, 연락처 등을 개인 정보를 포함하였습니다. 이번 프로젝트에서는

편의점이라는 특성상, **비회원 고객**이 전체 구매자의 다수를 차지할 수도 있으므로, 고객 정보는 선택적으로 등록될 수 있게 설계되었습니다.

거래는 고객 없이도 발생할 수 있도록 되어 있어, **비회원 트랜잭션**도 시스템에 기록 가능합니다.

- **Membership**

고객의 멤버십 상태 및 등급(GOLD, SILVER)을 관리하는 엔티티입니다.

각 **Membership**은 한 명의 고객에 대응하며, `level`, `join_date`, `discount_rate` 등의 속성을 가집니다. 이를 통해 고객별 혜택을 다르게 적용하거나, 등급별 고객의 구매 패턴을 분석해 상품 추천, 혜택 설계, 로열티 프로그램 등에 활용할 수 있습니다. Customer 과는 1:1 identifying relationship으로 연결됩니다.

- 주요 attribute 설명

`level` : 등급을 나타내며, GOLD, SILVER 구분됩니다.

`join_date` : 멤버십 가입일입니다.

`discount_rate` : 해당 등급에 부여된 할인 비율입니다.

- **변동 사항**

▪ 각 **Membership**은 한 명의 **Customer**에 정확히 1:1로 대응되므로, Customer가 존재해야 Membership이 존재할 수 있습니다.

▪ **1:1 Relationship**으로 설계하고, `customer_id` 를 기본키로 사용하면서 동시에 `Customer` 테이블로부터 외래키(FK)로 참조받습니다.

- **SaleTransaction**

**SaleTransaction**은 편의점에서 실제 결제가 발생한 **하나의 거래 단위**를 나타내는 엔티티입니다.

`saleTransaction_id` 로 식별되며, 거래 발생 시점의 날짜 및 시간, 결제 수단 등의 정보를 포함합니다.

거래는 반드시 `store_id`와 연결되어 어떤 매장에서 발생했는지를 명확히 하며, 옵션으로 `customer_id`가 포함되어 고객 기반 분석도 가능하게 합니다.

본 프로젝트에서는 정규화 원칙을 준수하기 위하여 할인 총액, 최종 결제 금액 등 계산으로 유도 가능한 파생 속성은 **SaleTransaction** 엔티티에 포함하지 않았습니다.

**SaleTransaction**은 특정 매장에서 발생하며(`store_id`), 선택적으로 특정 고객(`customer_id`)과 연결될 수 있습니다. 또한 하나의 거래는 여러 개의 상품 항목을 포함하므로, **SaleDetail**과 1:N 관계로 연결됩니다.

- **SaleDetail**

**SaleDetail** 엔티티는 SaleTransaction과 Product 간의 구체적인 구매 항목을 나타냅니다.

즉, 하나의 거래에 어떤 제품이 몇 개나 포함되었는지를 표현하는 역할로, 복합키 `(saleTransaction_id, upc)`를 가집니다.

속성으로는 `quantity`, `unit_price`, `discount_rate` 가 있으며,

각 항목별 실제 판매 수량, 단가, 할인율을 개별적으로 관리할 수 있도록 구성되어 있습니다.

## 2. Normalization Analysis

릴레이션이 BCNF를 만족하려면, 다음 두 가지 조건을 모두 만족

① 릴레이션이 최소한 1NF를 만족할 것

② 릴레이션에 존재하는 모든 함수 종속성  $X \rightarrow Y$ 에 대해:

- **X가 슈퍼키 (X가 릴레이션의 전체 속성 집합을 결정할 수 있어야 한다).**

- **최종 Functional Dependency Table**

릴레이션	속성 목록	함수 종속성 (FD)
Store	<u>store_id</u> , name, address, open_time, close_time, ownership_type	<u>store_id</u> → name, address, open_time, close_time, ownership_type → 매장은 store_id로 유일하게 결정됨
Vendor	<u>vendor_id</u> , name, phone_number, email	<u>vendor_id</u> → name, phone_number, email → 공급업체는 vendor_id로 유일하게 결정됨
Product	<u>upc</u> , name, brand, package_type, size, price, category, vendor_id	<u>upc</u> → name, brand, package_type, size, price, category, vendor_id → 상품은 upc로 유일하게 결정됨
Inventory	<u>store_id</u> , <u>upc</u> , quantity, reorder_threshold, reorder_amount	( <u>store_id</u> , <u>upc</u> ) → quantity, reorder_threshold, reorder_amount → 매장별 재고는 매장+상품 조합으로 결정됨
Customer	<u>customer_id</u> , name, phone_number, email	<u>customer_id</u> → name, phone_number, email → 고객은 customer_id로 유일하게 결정됨
Membership	<u>customer_id</u> , level, join_date, discount_rate	<u>customer_id</u> → level, join_date, discount_rate → 고객별 멤버십 1:1 관계
SaleTransaction	<u>saleTransaction_id</u> , transaction_time, payment_method, store_id, customer_id	<u>saleTransaction_id</u> → transaction_time, payment_method, store_id, customer_id → 거래는 고유 거래번호로 결정됨
SaleDetail	<u>saleTransaction_id</u> , <u>upc</u> , quantity, unit_price, discount_rate	( <u>saleTransaction_id</u> , <u>upc</u> ) → quantity, unit_price, discount_rate → 각 거래의 각 상품은 복합키로 결정됨

- 변경 사항

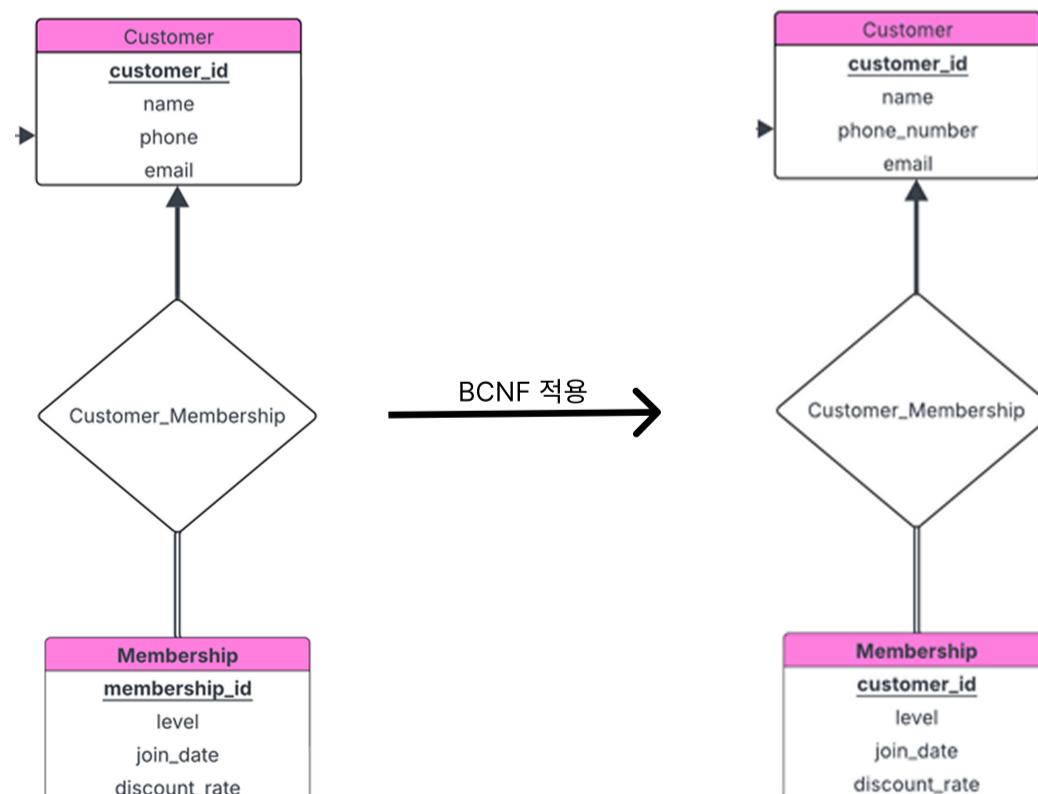


그림 2. BCNF 적용 전 후 비교 사진

- **BCNF 적용 전:**

- 초기 Logical Schema를 설계할 때, 모든 테이블에는 항상 별도의 id 속성(PK)가 있어야 한다고 판단하여 각 엔티티마다 고유 식별자를 만들었습니다. 이 과정에서 **Membership** 테이블에도 `membership_id`라는 독립적인 PK 속성을 생성하였습니다. 그러나 Membership 엔티티의 속성 구조를 살펴보면
- 속성: `membership_id, customer_id, level, join_date, discount_rate`
  - 주요 함수 종속성:
    - `customer_id → level, join_date, discount_rate`
    - `membership_id → customer_id`

`membership_id` 가 PK이지만, 실제로 멤버십 속성(`level`, `join_date`, `discount_rate`)은 `customer_id`에 의해 결정되는 것을 파악하였습니다. **transitive dependency**이 존재하게 되어 BCNF 위반 가능성이 있다고 판단하였습니다.

#### ■ 문제점:

- 하나의 고객은 단 하나의 멤버십만을 가질 수 있는데도 불구하고, 별도의 `membership_id`를 부여함으로써 중복 및 무결성 위반 결과적으로 Membership 릴레이션은 엄밀히 BCNF를 만족하지 않는 상태였습니다.

## 2-1. Step-by-Step BCNF Decomposition Process( Membership Entity)

### BCNF 위반 원인:

- 기존 스키마:
  - Membership** (`membership_id(PK)`, `customer_id(FK)`, `level`, `join_date`, `discount_rate`)
- 함수 종속성:

`customer_id → level, join_date, discount_rate`  
`membership_id → customer_id`

- `membership_id` 가 PK인데 실제 속성 결정은 `customer_id`에 의존 → **transitive dependency** 발생.

### BCNF 정규화 적용

- `membership_id` 제거
- `customer_id` 를 PK로 설정
- 최종 스키마:
  - Membership** (`customer_id(PK,FK)`, `level`, `join_date`, `discount_rate`)

### 다른 릴레이션

- Store, Vendor, Product, Inventory, Customer, SaleTransaction, SaleDetail 테이블은 최초 설계에서 이미 함수 종속성이 PK 기반으로 설계되어 있어 추가 분해 불필요.
- 각각 단일 속성 PK 또는 복합 PK를 이용해 모든 속성을 결정 → BCNF 자동 만족.

## 3. Physical Implementation

- 테이블 별 정리 Table

테이블	속성명	데이터 타입	제약조건	상세 설명
<b>Store</b>	store_id	INT	PRIMARY KEY	매장 고유 식별번호
	name	VARCHAR(50)	NOT NULL	매장명
	address	VARCHAR(200)	NOT NULL	매장 주소
	open_time	TIME	NOT NULL	영업 시작 시간
	close_time	TIME	NOT NULL	영업 종료 시간
	ownership_type	ENUM('Franchise','Corporate')	NOT NULL	매장 소유 유형
<b>Vendor</b>	vendor_id	INT	PRIMARY KEY	공급업체 고유 식별번호
	name	VARCHAR(50)	NOT NULL	공급업체명
	phone_number	VARCHAR(20)	NOT NULL	공급업체 연락처
	email	VARCHAR(100)	NOT NULL	공급업체 이메일
<b>Product</b>	upc	CHAR(12)	PRIMARY KEY	제품 식별을 위한 국제 표준 바코드
	name	VARCHAR(50)	NOT NULL	상품명
	brand	VARCHAR(50)	NOT NULL	브랜드명

테이블	속성명	데이터 타입	제약조건	상세 설명
	package_type	VARCHAR(50)	NOT NULL	포장 형태
	size	VARCHAR(50)	NOT NULL	용량 또는 크기
	price	DECIMAL(10,2)	NOT NULL	상품 가격
	category	VARCHAR(50)	NOT NULL	상품 분류
	vendor_id	INT	FOREIGN KEY → Vendor(vendor_id), ON DELETE SET NULL	공급업체 연계
<b>Inventory</b>	store_id	INT	FOREIGN KEY → Store(store_id), ON DELETE CASCADE	재고의 매장 소속
	upc	CHAR(12)	FOREIGN KEY → Product(upc), ON DELETE CASCADE	재고의 상품
	quantity	INT	NOT NULL	현재 재고 수량
	reorder_threshold	INT	NOT NULL	재주문 임계치
	reorder_amount	INT	NOT NULL	재주문 수량
	(store_id, upc)	(복합키)	PRIMARY KEY	매장-상품 복합키로 유일성 보장
<b>Customer</b>	customer_id	INT	PRIMARY KEY	고객 고유 식별번호
	name	VARCHAR(50)	NOT NULL	고객명
	phone_number	VARCHAR(20)	NOT NULL	연락처
	email	VARCHAR(100)	NOT NULL	이메일
<b>Membership</b>	customer_id	INT	PRIMARY KEY, FOREIGN KEY → Customer(customer_id), ON DELETE CASCADE	고객 1:1 멤버십 연계
	level	ENUM('Silver','Gold')	NOT NULL	멤버십 등급
	join_date	DATE	NOT NULL	가입일
	discount_rate	DECIMAL(5,2)	NOT NULL	할인율 (%)
<b>SaleTransaction</b>	saleTransaction_id	INT	PRIMARY KEY	거래 고유 식별번호
	transaction_time	DATETIME	NOT NULL	거래 발생 시각
	payment_method	ENUM('BankTransfer','Cash','CreditCard','MobilePayment')	NOT NULL	결제 수단
	store_id	INT	FOREIGN KEY → Store(store_id), ON DELETE SET NULL	거래 발생 매장
	customer_id	INT	FOREIGN KEY → Customer(customer_id), ON DELETE SET NULL	거래 고객 (선택적 연결)
<b>SaleDetail</b>	saleTransaction_id	INT	FOREIGN KEY → SaleTransaction(saleTransaction_id), ON DELETE CASCADE	거래 연결
	upc	CHAR(12)	FOREIGN KEY → Product(upc), ON DELETE RESTRICT	상품 연결 (삭제 제한)
	quantity	INT	NOT NULL	구매 수량
	unit_price	DECIMAL(10,2)	NOT NULL	거래 시점 판매 단가
	discount_rate	DECIMAL(5,2)	NOT NULL	거래 시점 적용 할인율
	(saleTransaction_id, upc)	(복합키)	PRIMARY KEY	거래-상품 복합키

- Unique & Not Null**

- 모든 PK는 자동으로 UNIQUE + NOT NULL 제약 조건이 걸립니다.
- 추가로 모든 속성에 NOT NULL을 설정하여 데이터 입력 시 필수 정보를 누락하는 것을 방지하였습니다.

속성	제약조건
모든 PK 속성	UNIQUE, NOT NULL
name, address, phone_number 등	NOT NULL

- ENUM**

- 값의 범위가 정의된 속성은 ENUM 자료형을 사용하여 입력 값을 사전에 제한하였습니다. 이를 통해 잘못된 데이터 입력을 원천적으로 방지할 수 있습니다. 또한 ENUM을 사용함으로 코드 가독성을 높이고, 값의 종류가 명확히 정의되어 있으므로 유지보수 측면에서 유리하다고 판단하였습니다.

- store.ownership\_type은 `Franchise`, `Corporate` 만 허용됩니다.
- membership.level은 `Silver`, `Gold` 만 허용합니다.
- SaleTransaction.payment\_method은 `BankTransfer`, `Cash`, `CreditCard`, `MobilePayment` 만 허용합니다.

속성	도메인	비즈니스 규칙 의미
Store.ownership_type	<code>Franchise</code> , <code>Corporate</code>	매장 형태 구분
Membership.level	<code>Silver</code> , <code>Gold</code>	멤버십 등급
SaleTransaction.payment_method	<code>BankTransfer</code> , <code>Cash</code> , <code>CreditCard</code> , <code>MobilePayment</code>	결제 수단

- Primary Key 제약 테이블

테이블	Primary Key
Store	<code>store_id</code>
Vendor	<code>vendor_id</code>
Product	<code>upc</code>
Inventory	( <code>store_id</code> , <code>upc</code> )
Customer	<code>customer_id</code>
Membership	<code>customer_id</code>
SaleTransaction	<code>saleTransaction_id</code>
SaleDetail	( <code>saleTransaction_id</code> , <code>upc</code> )

- Foreign Key 제약 테이블

### ON DELETE 동작 종류

#### 1. SET NULL :

- 부모 테이블 행 삭제 시 해당 테이블의 FK 값을 null로 변경합니다.
- 부모 삭제 후에도 자식 데이터를 유지하고 싶을 때 혹은 연관관계는 끊지만 기록을 보존하고 싶을 때 사용합니다.

#### 2. CASCADE

- 부모 테이블 행 삭제 시 자식 테이블의 해당 행도 함께 삭제합니다.
- 부모 삭제 시 자식도 의미가 없을 때 즉 완전한 종속 관계시 사용합니다.

#### 3. RESTRICT

- 부모 테이블의 행이 삭제될 수 없을 때 사용합니다.
- 자식 데이터를 보호해야 할 때 (무결성 위반 방지) 사용합니다.

외래키	참조 테이블	ON DELETE 동작	시나리오	상세 설명
<code>Product.vendor_id → Vendor</code>	<code>Vendor</code>	SET NULL	공급업체 삭제 시 상품은 유지 (판매 가능성 유지)	공급업체가 삭제되더라도, 해당 공급업체가 납품한 상품은 그대로 유지됨. 이미 입고된 상품은 매장에서 여전히 판매 가능. 이 때 <code>Product .vendor_id</code> 는 NULL로 변경되어 더 이상 특정 공급업체와 연결되지 않음.
<code>Inventory.store_id → Store</code>	<code>Store</code>	CASCADE	매장 삭제 시 해당 매장의 재고 자동 삭제	매장이 폐점되면, 해당 매장에 보관된 모든 재고는 의미가 없어짐. 따라서 매장 삭제 시 해당 매장의 재고 기록도 자동으로 삭제됨.
<code>Inventory.upc → Product</code>	<code>Product</code>	CASCADE	상품 삭제 시 관련 재고 자동 삭제	상품이 삭제되면, 더 이상 해당 상품을 매장에서 판매하거나 보관할 수 없음, 해당 상품과 연결된 모든 재고를 자동 삭제됨.
<code>SaleTransaction.store_id → Store</code>	<code>Store</code>	SET NULL	매장 삭제 후에도 거래 기록 유지	매장이 삭제되더라도 과거 발생한 <code>SaleTransaction</code> 은 보존됨. 과거 매출 통계 및 회계 기록을 유지할 수 있음. 이 때 거래기록에서 <code>store_id</code> 는 NULL로 변경됨.
<code>SaleTransaction.customer_id → Customer</code>	<code>Customer</code>	SET NULL	고객 삭제 후에도 거래 기록 유지	고객이 삭제되어도 기존 거래 내역은 삭제하면 안됨. 이 때 <code>SaleTransaction</code> 의 <code>customer_id</code> 는 NULL 처리됨.
<code>SaleDetail.saleTransaction_id → SaleTransaction</code>	<code>SaleTransaction</code>	CASCADE	거래 삭제 시 해당 상세내역도 자동 삭제	거래가 삭제되면, 해당 거래에 포함된 <code>SaleDetail</code> 들도 자동으로 삭제됨. 거래 자체가 사라지면 상세 항목도 의미가 없어지므로 같이 삭제.

외래키	참조 테이블	ON DELETE 동작	시나리오	상세 설명
SaleDetail.upc → Product	Product	RESTRICT	판매 기록이 존재하면 상품 삭제 불가	특정 상품이 과거 거래 내역에 기록되어 있을 경우, 해당 상품은 삭제할 수 없음. 판매 이력이 존재하는 상품 삭제 시 데이터 정합성 위반 가능성.
Membership.customer_id → Customer	Customer	CASCADE	고객 삭제 시 멤버십도 함께 삭제	고객이 삭제될 경우 해당 고객의 Membership 정보도 함께 삭제됨. 고객이 존재하지 않으면 멤버십 정보도 존재할 수 없음

- **Index 구현**

- DB 검색 성능 향상을 위해 자주 조회되는 속성에 대하여 추가로 인덱스를 생성하였습니다.
- 주어진 쿼리 7개를 기준으로 선정한 인덱스입니다.

인덱스명	대상 컬럼	상세 용도 및 기대효과
<b>idx_transaction_time</b>	SaleTransaction.transaction_time	거래 기록을 조회할 때 특정 기간 내 거래내역을 검색하는 경우가 빈번함 (예: 월별 매출, 분기별 실적 등). 이 인덱스를 통해 WHERE transaction_time BETWEEN ... 또는 ORDER BY transaction_time 과 같은 시간 기반 쿼리의 검색 속도를 크게 향상시킬 수 있음.
<b>idx_product_vendor</b>	Product.vendor_id	특정 공급업체가 납품하는 상품 목록을 조회하는 경우 사용됨. 예를 들어, 특정 Vendor가 납품하는 전체 제품군을 확인하거나 공급업체별 매출 분석을 수행할 때 매우 유용. 조인 연산 수행 시도 빠르게 처리됨 ( JOIN Product ON Vendor.vendor_id = Product.vendor_id ).
<b>idx_store_ownership</b>	Store.ownership_type	매장 유형(Franchise, Corporate)별로 매장 집계, 비교, 분석을 자주 수행하는 시나리오에 사용됨. 예를 들어, 프랜차이즈 매장과 직영 매장의 매출 비교, 상품 구색 차이 분석 등에서 해당 인덱스가 빠른 필터링을 가능하게 함 ( WHERE ownership_type = 'Franchise' ).

## 4. Application Development

C언어와 MySQL C API를 이용하여 DB와 연동하는 응용 프로그램을 개발하였습니다. 사용자가 메뉴를 통해 원하는 타입을 선택하여 쿼리를 수행하도록 구성하였습니다.

### 4-1. Database Connectivity Implementation Details

- MySQL C API로 MySQL 서버와 직접 통신하였습니다.
- 서버 연결 정보

```
const char *server = "localhost";
const char *user = "root";
const char *password = "1234";
const char *database = "store";
```

- MySQL 연결

- 프로그램 시작 시 mysql\_init()을 통해 핸들러를 초기화 한 뒤, SSL 모드를 비활성화하였습니다.
- mysql\_real\_connect()로 연결을 시도하였습니다.

```
MYSQL *conn = mysql_init(NULL);

if (conn == NULL) {
    fprintf(stderr, "mysql_init() failed\n");
    return 1;
}

unsigned int ssl_mode = SSL_MODE_DISABLED;
mysql_options(conn, MYSQL_OPT_SSL_MODE, &ssl_mode);

if (mysql_real_connect(conn, server, user, password, database, 0, NULL, 0) == NULL) {
    error_exit(conn);
}
```

### 4-2. Query Implementation

- 7가지 쿼리 정리 Table

TYPE	구현 내용	주요 기능 설명
<b>TYPE 1: Product Availability</b>	특정 상품의 재고 보유 매장 조회	특정 상품을 제품명, 브랜드, UPC 기준으로 입력받아 보유 매장과 재고 수량 출력
<b>TYPE 2: Top Selling Items</b>	매장별 최다 판매 상품 조회	최근 한 달간(과제 기간 기준 : 5월) 매장별 판매량 기준으로 최다 판매 상품 조회
<b>TYPE 3: Store Performance</b>	분기별 최고 매출 매장 조회	현재 분기(2분기) 내 가장 높은 매출을 기록한 매장 출력
<b>TYPE 4: Vendor Statistics</b>	최대 판매 공급업체 조회	각 공급업체별 총 판매수량 및 공급상품 수량 집계
<b>TYPE 5: Inventory Reorder Alerts</b>	재주문 필요 상품 조회	재고가 재주문 임계치 이하로 떨어진 상품 출력
<b>TYPE 6: Customer Purchase Patterns</b>	커피(Category)와 함께 많이 구매하는 상품 조회	로열티 고객이 커피(Category) 구매 시 함께 구매하는 Top 3 상품 출력
<b>TYPE 7: Franchise Comparison</b>	프랜차이즈와 직영매장 간 상품 다양성 비교	두 소유 유형별로 가장 다양한 상품 보유 매장 출력

- 공통 쿼리 실행 함수 : executeQuery()
  - 쿼리 실행, 결과 출력, 예외 처리를 공통화하여 중복을 최소화하였습니다.

```

void executeQuery(MYSQL *conn, const char *query) {
    if (mysql_query(conn, query)) {
        error_exit(conn);
    }
    MYSQL_RES *res = mysql_store_result(conn);

    // 예외 처리
    if (res == NULL) {
        printf("Query failed.\n");
        return;
    }

    if (mysql_num_rows(res) == 0) {
        printf("No result.\n");
        mysql_free_result(res);
        return;
    }

    int num_fields = mysql_num_fields(res);
    MYSQL_FIELD *fields = mysql_fetch_fields(res);

    // 컬럼명 출력
    for (int i = 0; i < num_fields; i++) {
        printf("%-20s", fields[i].name);
    }
    printf("\n");

    // 구분선 출력
    for (int i = 0; i < num_fields; i++) {
        printf("-----");
    }
    printf("\n");

    MYSQL_ROW row;
    while ((row = mysql_fetch_row(res))) {
        for (int i = 0; i < num_fields; i++) {
            printf("%-20s", row[i] ? row[i] : "NULL");
        }
        printf("\n");
    }

    mysql_free_result(res);
}

```

- TYPE 1
  - 사용자가 입력한 특정 상품을 기준으로 해당 상품을 보유하고 있는 매장 목록을 조회합니다.
  - 검색 기준으로 **UPC, 상품명, 브랜드명** 중 하나를 선택하여 조회할 수 있습니다.

- SQL 로직 설명

- `Store`, `Inventory`, `Product` 세 테이블을 `JOIN`
- 사용자가 입력한 검색 기준 (`upc`, `name`, `brand`)을 기준으로 `Product` 테이블을 필터
- 대소문자 구분을 피하기 위해 `LOWER()` 함수
- LIKE 연산을 통해 부분 검색 가능

```

SELECT
    s.name AS StoreName,
    p.name AS ProductName,
    p.brand,
    i.quantity
FROM
    Store s
JOIN
    Inventory i ON s.store_id = i.store_id
JOIN
    Product p ON i.upc = p.upc
WHERE
    LOWER(p.[검색필드]) LIKE LOWER('%[검색어]%');

```

- TYPE 2

- 각 매장에서 지난 한 달 동안 가장 많이 판매된 상품을 출력합니다.

- SQL 로직 설명:

- `SaleTransaction` 과 `SaleDetail` 을 조인하여 거래별 판매 데이터를 결합
- 판매수량(`quantity`) × 단가(`unit_price`) × (1 - 할인율/100)을 곱해 실제 순매출을 계산
- `WHERE` 절에서 현재 분기의 거래만 필터링(`QUARTER()` 와 `YEAR()` 함수를 활용)
- `GROUP BY` 를 통해 매장별로 총 매출을 집계하고, 가장 매출이 높은 매장 1개만 출력

```

WITH SalesPerProductPerStore AS (
    SELECT
        st.store_id,
        sd.upc,
        SUM(sd.quantity) AS volume
    FROM
        SaleTransaction st
    JOIN
        SaleDetail sd ON st.saleTransaction_id = sd.saleTransaction_id
    GROUP BY
        st.store_id, sd.upc
)
SELECT
    s.store_id AS StoreID,
    s.name AS StoreName,
    p.upc AS UPC,
    p.name AS ProductName,
    sq.volume AS Volume
FROM
    SalesPerProductPerStore sq
JOIN
    Store s ON s.store_id = sq.store_id
JOIN
    Product p ON p.upc = sq.upc
WHERE
    sq.volume = (
        SELECT
            MAX(sub.volume)
        FROM (
            SELECT
                st2.store_id,

```

```

        sd2.upc,
        SUM(sd2.quantity) AS volume
    FROM
        SaleTransaction st2
    JOIN
        SaleDetail sd2 ON st2.saleTransaction_id = sd2.saleTransaction_id
    WHERE
        st2.store_id = sq.store_id
    GROUP BY
        st2.store_id, sd2.upc
    ) AS sub
)
ORDER BY
    s.store_id;

```

- TYPE 3

- SQL 로직 설명
  - `SaleTransaction` 과 `SaleDetail` 을 조인하여 거래별 판매 데이터를 결합
  - 매출 계산 시, 판매수량(`quantity`) × 단가(`unit_price`) × (1 - 할인율/100)을 곱해 실제 순매출을 계산
  - `WHERE` 절에서 현재 분기의 거래만 필터링하기 위해 `QUARTER()` 와 `YEAR()` 함수
  - `GROUP BY` 를 통해 매장별로 총 매출을 집계하고, 가장 매출이 높은 매장 1개만 출력

```

SELECT
    s.store_id,
    s.name AS StoreName,
    SUM(sd.quantity * sd.unit_price * (1 - sd.discount_rate / 100)) AS total_revenue
FROM
    SaleTransaction st
JOIN
    SaleDetail sd ON st.saleTransaction_id = sd.saleTransaction_id
JOIN
    Store s ON st.store_id = s.store_id
WHERE
    QUARTER(st.transaction_time) = QUARTER(CURDATE())
    AND YEAR(st.transaction_time) = YEAR(CURDATE())
GROUP BY
    s.store_id, s.name
ORDER BY
    total_revenue DESC
LIMIT 1;

```

- TYPE 4

- SQL 로직 설명
  - `Vendor`, `Product`, `SaleDetail` 테이블을 조인
  - 각 공급업체가 공급한 상품 수량(`COUNT(p.upc)`)과 해당 상품의 총 판매수량(`SUM(sd.quantity)`) 집계
  - `GROUP BY` 로 공급업체별 집계를 수행한 후, 가장 많이 판매한 업체 1개를 `ORDER BY total_sold DESC LIMIT 1` 로 출력

```

SELECT
    v.vendor_id,
    v.name AS VendorName,
    COUNT(p.upc) AS total_products,
    SUM(sd.quantity) AS total_sold
FROM
    Vendor v
JOIN
    Product p ON v.vendor_id = p.vendor_id
JOIN
    SaleDetail sd ON p.upc = sd.upc
GROUP BY
    v.vendor_id, v.name
ORDER BY

```

```
total_sold DESC  
LIMIT 1;
```

- TYPE 5

- SQL 로직 설명

- `Inventory`, `Store`, `Product` 를 조인
    - `WHERE i.quantity < i.reorder_threshold` 조건으로 현재 재고가 재주문 임계치보다 작은 상품만 필터링
    - 매장 ID, 매장명, 상품 정보, 현재 재고, 임계치 정보를 출력

```
SELECT  
    s.store_id,  
    s.name AS StoreName,  
    p.upc,  
    p.name AS ProductName,  
    i.quantity,  
    i.reorder_threshold  
FROM  
    Inventory i  
JOIN  
    Store s ON i.store_id = s.store_id  
JOIN  
    Product p ON i.upc = p.upc  
WHERE  
    i.quantity < i.reorder_threshold;
```

- TYPE 6

- SQL 로직 설명:

- `Membership` 에 가입한 고객(`m.level IS NOT NULL`)만 대상
    - 거래 내역 중 서브쿼리에서 `p2.category = 'Coffee'` 를 이용하여 `saleTransaction_id` 를 추출
    - 거래 안에서 커피 외 다른 상품(`p.category <> 'Coffee'`)의 구매 수량을 집계
    - `SUM(sd.quantity)` 를 기준으로 내림차순 정렬
    - 가장 자주 함께 구매된 Top 3 상품을 출력

```
SELECT  
    p.name AS ProductName,  
    SUM(sd.quantity) AS total_quantity  
FROM  
    SaleTransaction st  
JOIN  
    SaleDetail sd ON st.saleTransaction_id = sd.saleTransaction_id  
JOIN  
    Product p ON sd.upc = p.upc  
JOIN  
    Membership m ON st.customer_id = m.customer_id  
WHERE  
    m.level IS NOT NULL  
    AND st.saleTransaction_id IN (  
        SELECT  
            sd2.saleTransaction_id  
        FROM  
            SaleDetail sd2  
        JOIN  
            Product p2 ON sd2.upc = p2.upc  
        WHERE  
            p2.category = 'Coffee'  
    )  
    AND p.category <> 'Coffee'  
GROUP BY  
    p.name  
ORDER BY
```

```
total_quantity DESC  
LIMIT 3;
```

- TYPE 7

- SQL 로직 설명:

- `Store` 와 `Inventory` 를 조인
    - 프랜차이즈( `ownership_type = 'Franchise'` )와 직영( `ownership_type = 'Corporate'` ) 매장을 각각 따로 집계
    - 각 매장 유형별로 가장 다양한 상품을 보유한 매장 1개씩을 출력
    - 두 결과를 `UNION ALL` 로 합쳐 결과로 출력

```
(  
    SELECT  
        s.store_id,  
        s.ownership_type,  
        s.name AS StoreName,  
        COUNT(DISTINCT i.upc) AS product_variety  
    FROM  
        Store s  
    JOIN  
        Inventory i ON s.store_id = i.store_id  
    WHERE  
        s.ownership_type = 'Franchise'  
    GROUP BY  
        s.store_id, s.name  
    ORDER BY  
        product_variety DESC  
    LIMIT 1  
)  
UNION ALL  
(  
    SELECT  
        s.store_id,  
        s.ownership_type,  
        s.name AS StoreName,  
        COUNT(DISTINCT i.upc) AS product_variety  
    FROM  
        Store s  
    JOIN  
        Inventory i ON s.store_id = i.store_id  
    WHERE  
        s.ownership_type = 'Corporate'  
    GROUP BY  
        s.store_id, s.name  
    ORDER BY  
        product_variety DESC  
    LIMIT 1  
);
```

### 4-3. Error Handling and User Interface Design

- User Interface Design

- 1. 프로그램 시작 화면

- a. 프로그램 실행 시 아래와 같은 메인 메뉴가 반복 출력됩니다.
    - b. 사용자는 타입 번호를 입력하여 원하는 결과를 출력할 수 있습니다.
    - c. 메뉴는 무한 루프를 통해 반복되어 사용자가 여러 쿼리를 연속으로 실행할 수 있도록 구현하였습니다.

```

PS C:\Users\hanna\Desktop\dbhw> ./main.exe

----- 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

```

2. 타입별 실행 결과

a. TYPE 1

```

----- 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: 1
----- TYPE 1 -----
** Which stores currently carry a certain product (by UPC,name, or brand), and how much inventory do they have? **
Enter product : coffee
Enter product identifier: name
StoreName      ProductName      brand      quantity
-----
Central Market    Coffee        Coffee Master    2

```

b. TYPE 2

```

5. TYPE 5
6. TYPE 6
7. TYPE 7
0. QUIT
Select: 2
----- TYPE 2 -----
**Which products have the highest sales volume in each store over the past month? **
StoreID      StoreName      UPC      ProductName      Volume
-----
1           Central Market  00000000001  Coffee          3
2           East Point     00000000003  Cheese Sandwich  3
3           West Mall      00000000004  Ham Sandwich   2
4           North Plaza    00000000005  Orange Juice    5
5           South Corner   00000000006  Apple Juice    2
6           City Center   00000000007  Potato Chips   2
7           Harbor Side   00000000008  Chocolate Muffin 1
8           Airport Store  00000000009  Organic Milk    1
9           Parkview       00000000010  Frozen Pizza   1
10          University Gate 00000000011  Cappuccino     3
11          River Mall    00000000012  Bagel          1

```

c. TYPE 3

```

----- 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: 3
----- TYPE 3 -----
**Which store has generated the highest overall revenue this quarter? **
store_id      StoreName      total_revenue
-----
2             East Point    16.50000000

```

d. TYPE 4

```

----- 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: 4
----- TYPE 4 -----
** Which vendor supplies the most products across the chain, and how many total units have been sold? **
vendor_id      VendorName      total_products      total_sold
-----
6             Coffee Master    6                  10

```

e. TYPE 5

----- 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: 5					
----- TYPE 5 -----					
** Which products in each store are below the reorder threshold and need restocking? **					
store_id	StoreName	upc	ProductName	quantity	reorder_threshold
1	Central Market	00000000001	Coffee	2	10
1	Central Market	00000000002	Latte	5	10
1	Central Market	00000000003	Cheese Sandwich	4	10
1	Central Market	00000000005	Orange Juice	2	5
1	Central Market	00000000006	Apple Juice	0	5
1	Central Market	00000000007	Potato Chips	3	20
1	Central Market	00000000009	Organic Milk	3	10
2	East Point	00000000011	Cappuccino	3	5
2	East Point	00000000013	Yogurt	2	5
2	East Point	00000000014	Green Smoothie	1	5
3	West Mall	00000000015	Strawberry Smoothie	0	5
3	West Mall	00000000018	Sparkling Water	2	5
3	West Mall	00000000019	Energy Drink	3	5
3	West Mall	00000000020	Tofu Snack	4	5

f. TYPE 6

----- 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: 6					
----- TYPE 6 -----					
** List the top 3 items that loyalty program customers typically purchase with coffee. **					
ProductName	total_quantity				
Cheese Sandwich	2				
Chocolate Muffin	2				
Potato Chips	1				

g. TYPE 7

----- 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: 7					
----- TYPE 7 -----					
Among franchise-owned stores, which one offers the widest variety of products, and how does that compare to corporate-owned stores?					
store_id	ownership_type	StoreName	product_variety		
1	Franchise	Central Market	10		
2	Corporate	East Point	4		

### 3. 프로그램 종료

- 사용자가 0을 입력하면 MySQL 연결을 종료 후 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 terminated.					

- 화면 출력 코드

```
while (1) {
    printf("\n----- SELECT QUERY TYPES -----");
    printf("1. TYPE 1\n");
    printf("2. TYPE 2\n");
    printf("3. TYPE 3\n");
    printf("4. TYPE 4\n");
    printf("5. TYPE 5\n");
    printf("6. TYPE 6\n");
    printf("7. TYPE 7\n");
    printf("0. QUIT\n");
    Select: 0
    Program terminated.
```

```

printf("6. TYPE 6\n");
printf("7. TYPE 7\n");
printf("0. QUIT \n");

printf("Select: ");

int choice;
scanf("%d", &choice);

int ch;
while ((ch = getchar()) != '\n' && ch != EOF); // 버퍼 비우기

switch (choice) {
    case 1: executeType1(conn); break;
    case 2: executeType2(conn); break;
    case 3: executeType3(conn); break;
    case 4: executeType4(conn); break;
    case 5: executeType5(conn); break;
    case 6: executeType6(conn); break;
    case 7: executeType7(conn); break;
    case 0:
        mysql_close(conn);
        printf("Program terminated.\n");
        return 0;
    default:
        printf("Invalid choice.\n");
}
}
}

```

- 에러 처리
  - 모든 MySQL 함수 호출 시 에러 체크를 수행합니다.
  - 오류 발생 시 mysql\_error() 사용하여 상세 에러 메시지를 출력 후 프로그램을 종료시킵니다.

```

void error_exit(MYSQL *conn) {
    fprintf(stderr, "SQL error: %s\n", mysql_error(conn));
    mysql_close(conn);
    exit(1);
}

```

## 5. Testing and Validation

schema.sql

sample\_data.sql

위 파일을 기준으로 수행한 결과입니다. 데이터 무결성과 비즈니스 규칙이 정상적으로 적용되는지 검증하였습니다.

### 5-1 Test Case Description and Results

- TYPE 1 - Product Availability

**목표:** 상품명 검색을 통해 해당 상품 재고를 보유한 매장 목록 확인

**입력값:**

- 검색어 : "Coffee"
- 검색 필드 : name

**예상결과:**

- ProductName이 'Coffee'인 상품을 보유 중인 매장 목록 출력

```
----- TYPE 1 -----
** Which stores currently carry a certain product (by UPC,name, or brand), and how much inventory do they have? **
Enter product : coffee
Enter product identifier: name
StoreName      ProductName      brand      quantity
-----  

Central Market    Coffee        Coffee Master    2
```

**실제결과:** 예상과 일치 → 정상 동작

- TYPE 2 - Top Selling Items

**목표:** 지난 한 달간 각 매장별로 가장 많이 팔린 상품 확인

**예상결과 예시:**

```
----- TYPE 2 -----
**Which products have the highest sales volume in each store over the past month? **
StoreID      StoreName      UPC      ProductName      Volume
-----  

1           Central Market  00000000001  Coffee            3
2           East Point       00000000003  Cheese Sandwich   3
3           West Mall        00000000004  Ham Sandwich     2
4           North Plaza      00000000005  Orange Juice     5
5           South Corner     00000000006  Apple Juice      2
6           City Center      00000000007  Potato Chips     2
7           Harbor Side      00000000008  Chocolate Muffin 1
8           Airport Store     00000000009  Organic Milk     1
9           Parkview         00000000010  Frozen Pizza     1
10          University Gate  00000000011  Cappuccino      3
11          River Mall       00000000012  Bagel           1
```

**실제결과:** 예상과 일치 → 정상 동작

- TYPE 3 - Store Performance

**목표:** 현재 분기 매출 1위 매장 확인 (할인 적용)

**예상결과:**

- 현재 분기의 전체 SaleTransaction + SaleDetail을 통해 매출 계산 후 매출 1위 매장 출력
- 할인 적용된 총 매출 정상 계산됨

```
----- TYPE 3 -----
**Which store has generated the highest overall revenue this quarter? **
store_id      StoreName      total_revenue
-----  

2           East Point      16.50000000
```

**실제결과:** 정확한 매출 결과 출력 → 정상 동작

- TYPE 4 - Vendor Statistics

**목표:** 공급업체 중 가장 공급 하는 상품이 많은 공급 업체와 수량 확인

**예상결과 예시:**

```
----- TYPE 4 -----
** Which vendor supplies the most products across the chain, and how many total units have been sold? **
vendor_id      VendorName      total_products      total_sold
-----  

6           Coffee Master      6                  10
```

**실제결과:** 공급업체 집계 정상 출력 → 정상 동작

- TYPE 5 - Inventory Reorder Alerts

**목표:** 재주문 임계치 이하 상품 확인

**테스트 시나리오 설명:**

- Inventory 테이블에 임계치 미만 재고 설정 (`quantity < reorder_threshold`)

**예상결과 예시:**

----- TYPE 5 -----					
** Which products in each store are below the reorder threshold and need restocking? **					
store_id	StoreName	upc	ProductName	quantity	reorder_threshold
1	Central Market	00000000001	Coffee	2	10
1	Central Market	00000000002	Latte	5	10
1	Central Market	00000000003	Cheese Sandwich	4	10
1	Central Market	00000000005	Orange Juice	2	5
1	Central Market	00000000006	Apple Juice	0	5
1	Central Market	00000000007	Potato Chips	3	20
1	Central Market	00000000009	Organic Milk	3	10
2	East Point	00000000011	Cappuccino	3	5
2	East Point	00000000013	Yogurt	2	5
2	East Point	00000000014	Green Smoothie	1	5
3	West Mall	00000000015	Strawberry Smoothie	0	5
3	West Mall	00000000018	Sparkling Water	2	5
3	West Mall	00000000019	Energy Drink	3	5
3	West Mall	00000000020	Tofu Snack	4	5

실제결과: 부족 재고 리스트 정상 출력 → 정상 동작

- TYPE 6 - Customer Purchase Patterns

목표: 멤버십 고객이 커피(Category) 구매 시 동반 구매한 상품 Top 3 출력

예상결과 예시:

----- TYPE 6 -----	
** List the top 3 items that loyalty program customers typically purchase with coffee. **	
ProductName	total_quantity
Cheese Sandwich	2
Chocolate Muffin	2
Potato Chips	1

실제결과: 정상 동작

- TYPE 7 - Franchise Comparison

목표: 프랜차이즈와 직영 매장 각각 가장 다양한 상품 보유 매장 확인

입력값: 없음

예상결과 예시:

----- TYPE 7 -----			
Among franchise-owned stores, which one offers the widest variety of products, and how does that compare to corporate-owned stores?			
store_id	ownership_type	StoreName	product_variety
1	Franchise	Central Market	10
2	Corporate	East Point	4

실제결과: 매장유형별로 집계 → 정상 동작

## 6. 결론

본 프로젝트에서는 C언어와 MySQL을 활용하여 편의점 매장 관리 시스템을 설계하고 구현하였습니다. 데이터베이스 시스템의 전체 개발 프로세스를 경험하였습니다. 실습을 통해 직접 구현하고 검증한 것은 다음과 같습니다.

- 관계형 데이터베이스의 정규화 원칙 적용 (BCNF 완전 정규화)
- 복잡한 다중 JOIN 기반의 실무형 SQL 쿼리 작성
- 다양한 제약조건을 활용한 비즈니스 규칙 강제
- C언어와 MySQL 연동을 통한 응용 프로그램 개발
- 샘플 데이터를 기반으로 한 실제 시뮬레이션 및 테스트 검증

이번 프로젝트를 통해 데이터베이스 설계의 중요성과, 무결성 유지를 위한 제약조건을 직접 실습하며 깊이 이해할 수 있었습니다.