데이터베이스 프로젝트 2

20161277 김재훈

2025-06-13

Logical schema

이 logical schema 는 모든 relation 이 BCNF 를 만족하도록 설계되어 있습니다. 각 relation 에서 모든 non-trivial functional dependency 의 좌변이 superkey 가 되도록 구성되어 있으며, 각 entity 는 해당 entity 를 대표하는 primary key 를 갖습니다. 예를 들어, STORE 테이블에서 StoreID 는 superkey 이며, StoreID 가 다른 모든 속성(Name, Address, OpeningHours, OwnershipType)을 결정합니다. STORE, CUSTOMER, VENDOR, PRODUCT, TRANSACTION 은 각각 단일 attribute 를 primary key 로 사용하고, TRANSACTIONITEM 과 INVENTORY 는 composite primary key 를 사용하여 BCNF 를 달성했습니다.

TRANSACTION 테이블의 CustomerID 는 nullable 로 설계되어 비회원 고객의 거래를 허용합니다. CUSTOMER 테이블에는 오직 회원인 고객들만 저장되며, 비회원 고객은 데이터베이스에 저장되지 않습니다. 이는 모든 transaction 을 기록하면서도 고객 정보가 없는 일회성 구매를 허용하는 설계입니다.

Physical schema

이 physical schema 에는 데이터 무결성을 보장하기 위한 다양한 constraint 가 명시되었습니다. CHECK constraint 를 통해 Price, TotalAmount, Subtotal 은 0 이상의 값만 허용하고, Quantity 는 0 보다 큰 값만 허용하도록 제한했습니다. ENUM 타입을 사용하여 OwnershipType 과 PaymentMethod 의 값을 제한하고, UNIQUE constraint 를 통해 Customer 의 Email 중복을 방지했습니다. 또한 DEFAULT CURRENT_TIMESTAMP 를 사용하여 거래 시간을 자동으로 기록하도록 설정했습니다.

PRIMARY KEY 는 각 테이블의 고유 식별자를 정의하며, 화살표로 표시되는 foreign key 는 테이블 간의 관계를 설정합니다. Nullable 여부는 business rule 에 따라 분리되어 있습니다. 앞서 설명했듯 CustomerID 만 nullable 하게 지정되었으며, 나머지는 NOT NULL 을 유지합니다.

schema.sql

모든 foreign key constraint 가 referential integrity 를 유지하도록 설계되었습니다. TRANSACTION 은 반드시 유효한 STORE 에서 발생해야 하고, TRANSACTIONITEM 은 유효한 TRANSACTION 과 PRODUCT 를 참조해야 하며, INVENTORY 는 유효한 STORE 와 PRODUCT 의 조합을 참조해야 합니다. CustomerID 의 nullable 특성을 제외하고는 모든 foreign key 가 NOT NULL 로 설정되어 필수적인 참조 관계를 보장합니다.

CASCADE 는 부모가 변경될 때 자식도 함께 변경되어야 하는 관계에 사용되었습니다. CASCADE 는 부모 레코드의 변경(예: ID 업데이트)이 자식 레코드에 자동으로 반영되도록 하며, 자식 레코드가 부모 없이는 의미가 없을 때(예: 트랜잭션 없는 트랜잭션 아이템) 사용됩니다. 즉, TRANSACTION 이 삭제될 경우 관련 TRANSACTIONITEM 도 ON DELETE CASCADE 로 자동 삭제됩니다. 반면, RESTRICT 는 기록 데이터

보존이나 비즈니스 연속성 유지와 같이 중요한 데이터가 실수로 삭제되는 것을 방지하기 위해 사용됩니다 (예: 판매 내역이 있는 상점을 삭제할 수 없음).

sample_data.sql

샘플 데이터는 현실성 있는 분포와 다양한 시나리오를 반영하도록 구성되었습니다. 예를 들어, store 별 OwnershipType 분포, 다양한 제품군과 vendor 매핑, 고객 구매 이력의 다양성, transaction 의 날짜 분산 등을 통해 실제 운영 환경을 시뮬레이션할 수 있도록 설계되었습니다. 비회원/회원 고객을 함께 반영한 transaction 도 포함되어 있습니다.

main.cpp

main.cpp 는 각 쿼리 유형에 대해 함수를 정의하고, 사용자 입력을 받아서 동적 SQL 을 생성한 후 실행합니다.

쿼리 1: Which stores currently carry a product (by UPC, name, or brand), and how much inventory do they have?

INVENTORY entity 를 중심으로 STORE 와 PRODUCT entity 를 join 해서 해결합니다.

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

TRANSACTION 과 TRANSACTIONITEM entity 를 활용하되, 핵심은 각 store 별로 가장 많이 팔린 product 를 찾는 것입니다. 이를 위해 ROW_NUMBER() OVER (PARTITION BY s.StoreID ORDER BY SUM(ti.Quantity) DESC) window function 을 사용합니다. PARTITION BY s.StoreID 가 중요한 이유는 전체에서 가장 많이 팔린 product 가 아니라 각 store 마다 가장 많이 팔린 product 를 찾아야 하기 때문입니다. DATE_SUB(CURDATE(), INTERVAL 1 MONTH)를 통해 지난 달 데이터만 필터링하고, GROUP BY s.StoreID, ti.UPC 로 store-product combination 별로 quantities 를 sum 한 후, 최종적으로 WHERE rn = 1 로 각 store 의 top product 만 선별합니다.

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

QUARTER(t.DateTime) = QUARTER(CURDATE())와 YEAR(t.DateTime) = YEAR(CURDATE()) 조건을 통해 현재 year 의 현재 quarter 데이터만 필터링합니다. GROUP BY s.StoreID 로 store 별 aggregation 을 하고 SUM(t.TotalAmount)로 total revenue 를 계산한 후, ORDER BY DESC 와 LIMIT 1 로 highest revenue store 를 찾습니다.

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

VENDOR 와 PRODUCT 를 INNER JOIN 으로 연결해 vendor 별 product count 를 구하고, TRANSACTIONITEM 과는 LEFT JOIN 을 사용합니다. LEFT JOIN 을 사용하는 이유는 아직 한 번도 팔리지 않은 product 들도 포함해야 하기 때문입니다. COUNT(DISTINCT p.UPC)로 vendor 별 unique product 수를 세고, 최종적으로 ProductCount DESC, TotalUnitsSold DESC 로 정렬해 가장 많은 product 를 공급하는 vendor 를 찾습니다.

쿼리 5: Which products in each store are below reorder threshold and need restocking?

WHERE i.CurrentStock < i.ReorderThreshold 조건만으로 restocking 이 필요한 products 를 찾을 수 있습니다.

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

같은 TransactionID 를 공유하는 products 를 찾아야 합니다. ti1 을 coffee products 로, ti2 를 같은 transaction 의 다른 products 로 사용합니다. JOIN TRANSACTIONITEM ti1 ON t.TransactionID = ti1.TransactionID 와 JOIN TRANSACTIONITEM ti2 ON t.TransactionID = ti2.TransactionID 을 통해 같은 transaction 에 속한 product pairs 를 만들고, ti2.UPC != ti1.UPC 조건으로 coffee 자체는 제외합니다. t.CustomerID IS NOT NULL 조건이 회원인 고객만 필터링합니다. 최종적으로 GROUP BY p2.UPC 로 각 product 별로 quantities 를 sum 하고 상위 3 개를 선택합니다.

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

COUNT(DISTINCT i.UPC)로 각 store 의 product variety 를 계산하고, ROW_NUMBER() OVER (PARTITION BY s.OwnershipType ORDER BY COUNT(DISTINCT i.UPC) DESC)를 통해 ownership type 별로 랭킹을 만듭니다. 즉, Company-owned stores 끼리 ranking 하고 Franchise stores 끼리 따로 rank 합니다. WHERE rn = 1 으로 각 ownership type 에서 가장 다양한 products 를 가진 store 만 골라냅니다.