Genevieve Awa
HTTP5126 Final Project Report
Dec 11, 2024

Perfume Database Proposal

1. Real World Scenario (2 marks)

As a database manager hired by a retail perfume company to manage their database system, which tracks brands, products, notes (fragrance components), sales, and related information, your role is to ensure seamless product updates and enable effective decision-making through inventory management, sales analysis, and marketing strategies.

2. Problems & Features (6 marks)

Problem 1: Product Inventory Management

Relevant Tables: product, brand, retail, note, product_retail

Feature: Stored Procedure and Trigger

Solution:

Managing inventory requires handling frequent product updates (e.g., new products, discontinued items, or name changes) while maintaining consistency. A stored procedure, **usp_add_product_brand_note_retail**, automates the addition or update of products, their brands, fragrance notes, and retail availability. By consolidating multiple steps into one, this procedure minimizes errors and ensures consistency across tables.

Additionally, a trigger, **trg_product_update**, automatically logs product name changes in the audit_log table, providing traceability and accountability. This ensures that inventory data remains accurate and auditable for decision-making and compliance.

Problem 2: Fragrance Analysis for Marketing

Relevant Tables: product, sale, retail, note, product_retail

Feature: View and Function

Solution:

The **fragrance_profile_view** consolidates product details (e.g., name, fragrance notes, price, and brand) into an easily accessible format, enabling marketers to identify popular notes or trends across brands.

A function, **fn_get_total_profit,** calculates product profitability by considering revenue and cost price. This helps identify high-performing products and supports financial planning.

3. Architecture Description (16 marks)

Database Name: perfume_db

Database Tools (3 marks) & Justification (3 marks)

View: fragrance_profile_view

Combines multiple tables (product, note, retail, and brand) into a unified format. It simplifies queries for marketers, enabling quick analysis of fragrance profiles and pricing trends.

Trigger: trg_product_update

The trigger logs updates to product information in the audit_log table, ensuring accountability and traceability in product management. Specifically, when a product's name is changed, the trigger captures the product_id, the old name, the new name, and the timestamp of the change.

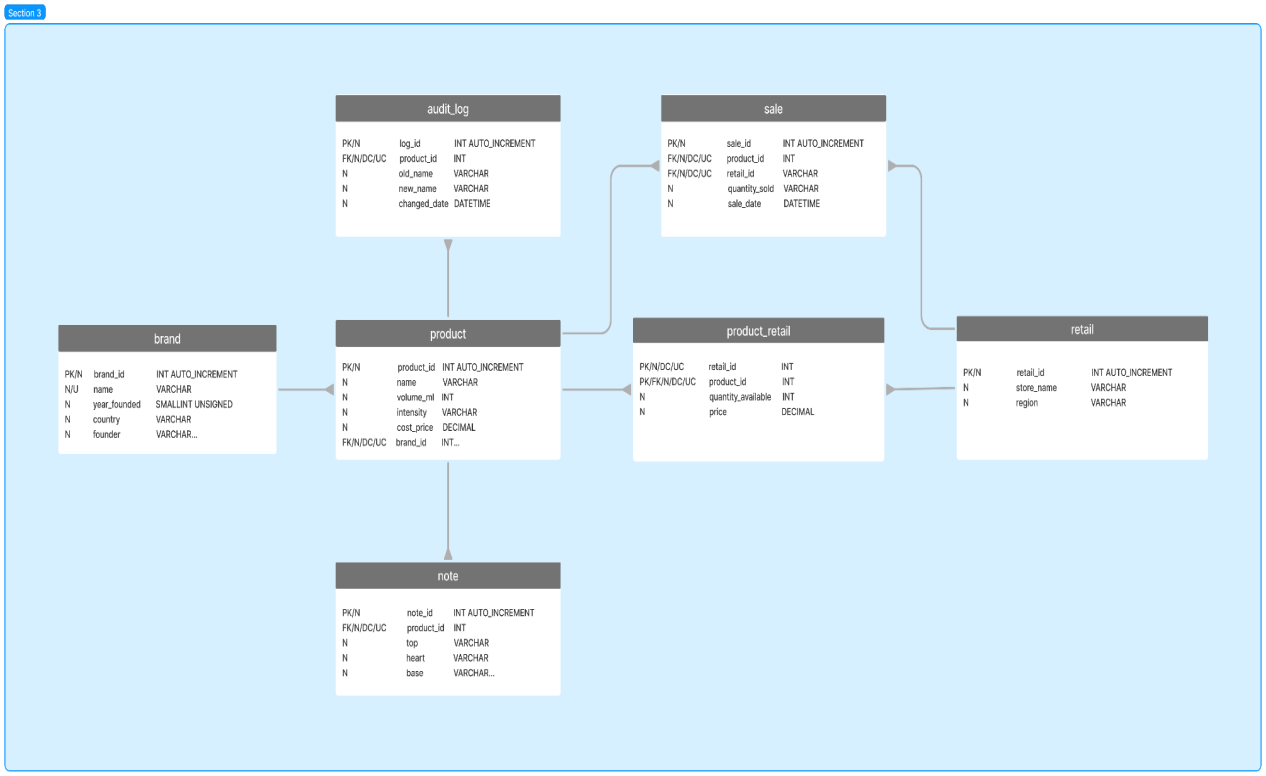
Procedure: usp_add_product_brand_note_retail

A stored procedure to add a new brand, product with associated note, and retail data. Instead of requiring multiple individual SQL statements to insert data into product, note, and brand, retail, and product_note, the procedure handles everything in one step. This reduces the potential for errors, ensures consistency across related tables, and improves efficiency for end users, particularly employees tasked with data entry.

Function: fn_get_total_profit

This function calculates profitability by subtracting cost from revenue. It provides insights into product performance, helping managers prioritize restocking and marketing efforts.

Database ERD Diagram (7 marks) & Justification (2 marks)



The Perfume Database schema is designed for efficient organization, flexibility, and normalization, enabling seamless management of brands, products, fragrance notes, retail availability, sales tracking, and audit logging. The brand table uniquely identifies each brand with a primary key (brand_id) and includes attributes such as name, year of establishment, and country. The product table connects to the brand table via a foreign key, storing details such as product name, volume, intensity, and cost price, establishing a one-to-many relationship between brands and products.

The note table categorizes fragrance components (top, heart, and base notes) with a unique note_id, directly linking each product to its unique fragrance profile in a one-to-many relationship. This replaces the earlier product_note table, simplifying the schema while preserving flexibility for detailed fragrance analysis. The retail table tracks store-specific attributes like store name and region, while the product_retail table manages the many-to-many relationship between products and stores, enabling dynamic pricing and regional inventory management with attributes such as price and quantity available.

The sales table captures transactional data, including products sold, quantities, and sale dates, allowing for detailed revenue and profitability analysis. This is complemented by the fn_get_total_profit function, which calculates the profitability of individual products by subtracting cost from revenue, offering critical insights for financial planning and restocking decisions.

The audit_log table enhances accountability by recording changes to product names, including product_id, old name, new name, and the timestamp of each update. This ensures traceability, compliance, and a historical record of changes, vital for maintaining data integrity over time.

The schema’s constraints, including primary keys, foreign keys, NOT NULL, ON UPDATE CASCADE, and ON DELETE CASCADE, ensure data consistency and integrity while supporting efficient database operations. These enhancements make the database highly scalable, user-friendly, and well-suited to the complex needs of a retail perfume business.

4. Retrospective

a. Changes from Proposal:

- A sales table was added to track transactions, enabling revenue and profit analysis.
- The product_note bridging table was replaced with a direct relationship between product and note.
- The retail table was normalized, and a product_retail table was introduced to handle the many-to-many relationship between products and stores.
- A function for calculating profit (fn_get_total_profit) was implemented, enhancing financial insights.

b. Challenges and Solutions

- Handling Many-to-Many Relationships: Normalizing the retail and product tables required adding a product_retail table. This resolved issues with data duplication and allowed dynamic pricing and inventory management.
- Avoiding Duplicate Data: The usp_add_product_brand_note_retail procedure was designed to check for existing entries before inserting new ones, preventing redundancy.
- Error Logging: Initial attempts to link product updates to the audit_log table caused foreign key conflicts. Adjusting the schema to include ON DELETE CASCADE resolved the issue.

- **Solved a real-world problem:**

The database now supports seamless inventory updates, accurate tracking of sales and profits, and efficient marketing analysis, making it a powerful tool for retail management.

c. Future Plans

Planned enhancements include adding Order, Supplier, and Customer tables to support end-to-end transaction tracking. A stored procedure for recommending products based on purchase history will enhance personalization, while triggers for low-stock alerts will improve inventory management. These features will make the database even more comprehensive, scalable, and aligned with business needs.