Genevieve Awa
HTTP5126 Final Project Proposal Overview
Nov 21, 2024

Perfume Database Proposal

Real World Scenario (2 marks)

As a database manger hired by a perfume company to manage their database system, which tracks brands, products, notes (fragrance components), and related information. The database will be used by managers, employees, and retail partners to ensure seamless data handling and availability of insights for decision-making, such as inventory management, marketing, and product innovation.

Problems & Features (6 marks)

Problem 1: Product Inventory Management

Relevant Tables: product, brand, retail

Managers need to quickly update product information (e.g., new products, discontinued products) while keeping track of brand associations.

Feature: Create a feature using SQL to easily insert, update, or delete products and their relationships to brands. This can include triggers to notify staff when changes are made.

Solution:

Managers often face the challenge of keeping the database updated with accurate product information, such as when new products are introduced or old ones are discontinued. To address this, the database provides a stored procedure called add_new_product. This procedure allows users to efficiently add new products and their associated fragrance notes in one operation. By automating this process, managers can ensure consistency and reduce errors. Additionally, a trigger called product_update_trigger is in place to log any updates made to the product table, capturing changes such as product name updates or volume modifications. This ensures that all modifications are traceable and auditable, providing transparency and accountability in inventory management.

Problem 2: Fragrance Analysis for Marketing

Relevant Tables: product_note, note

Marketing teams want to analyze the fragrance profile of products to identify trends in popular notes.

Feature: Build a SQL view that joins products with their associated fragrance notes, allowing quick access to top, heart, and base notes for each product.

Solution:

Marketing teams need to analyze and identify trends in fragrance profiles to design better advertising strategies and develop new products. To simplify this process, the database includes a view called fragrance_profile_view. This view consolidates data from multiple tables, providing a complete picture of each product's fragrance composition (top, heart, and base notes) along with its brand. With this view, marketers can query data such as all products containing a specific note or compare fragrance profiles across brands. For instance, they might use the view to identify which top notes are most popular among customers. This solution eliminates the need for marketing teams to write complex SQL joins, saving time and enabling them to focus on strategic decision-making. Furthermore, the function get_total_revenue_per_product can be used to calculate the total available product with price across all retail stores.

Architecture Description (16 marks)

Database Name: perfume_db

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

View: fragrance_profile_view

A view to display the complete fragrance profile for each product. This will consolidate complex data relationships into a single, simplified structure. It combines information about products, their associated brands, and the detailed fragrance notes (top, heart, and base). This is especially useful for marketing teams or product developers who need to analyze trends in fragrance profiles without diving into the complexities of multiple table joins. For example, a user can query the view to see all perfumes that include a specific top note or compare brands based on their fragrance profiles. Views ensure data abstraction, making it easier for non-technical users to access meaningful insights without needing to understand the underlying schema or write SQL queries themselves.

Trigger: product_update_trigger

A trigger to log updates to product inventory in a separate audit table. This will ensure accountability and traceability in product management. When product details (such as name) are updated, the trigger automatically logs the changes into an audit table. This feature is crucial for maintaining a history of updates, which can be referenced later for troubleshooting, compliance, or managerial review. For instance, if a product's name is changed inadvertently or without proper authorization, the audit log provides evidence of the change, including the old and new values and the timestamp. This automation reduces the manual effort required to maintain records and ensures consistency by capturing every update in real time.

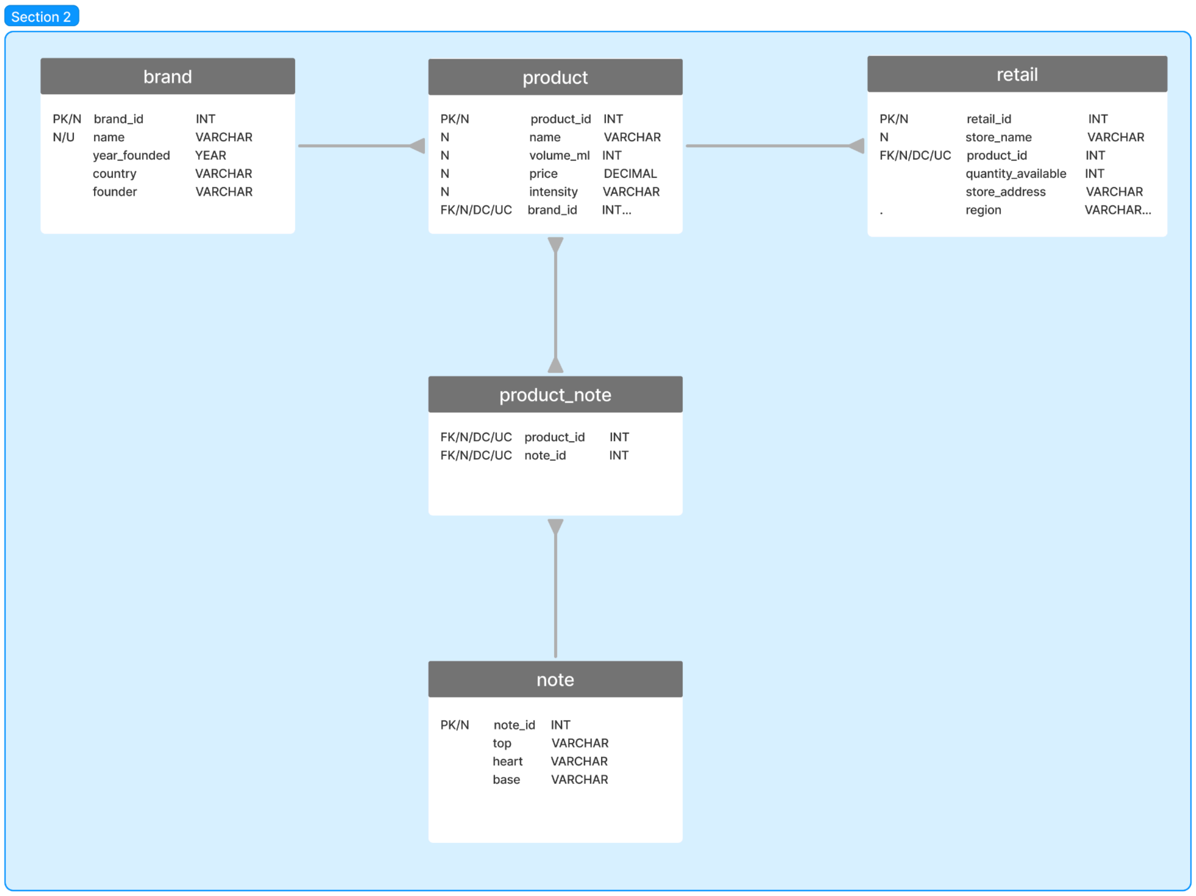
Procedure: add_new_product

A stored procedure to add new products with associated notes. This procedure simplifies the process of adding a new product and its associated notes in a single operation. Instead of requiring multiple individual SQL statements to insert data into product, note, 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. For example, when a new perfume is launched, the user only needs to call this procedure with the necessary details, and the product and its fragrance notes are seamlessly integrated into the database. This is particularly useful in a fast-paced retail environment where quick and accurate data updates are critical.

Function: get_total_revenue_per_product

This function calculates the total revenue potential of a specific product across all retail stores, which is crucial for financial planning and decision-making. It allows managers to quickly compute the total value of a product available in all retail locations. For example, if a product is stocked at varying quantities across multiple stores, this function aggregates the revenue potential based on the product's price and quantity available, providing a consolidated view of its financial worth. This insight can help managers prioritize restocking or redistribution of high-value products. Without this function, users would need to repeatedly write complex queries to compute revenue, but with the function, the calculation becomes standardized, reusable, and simplified for end-users.

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



brand

PK/N	brand_id	INT
N/U	name	VARCHAR
	year_founded	YEAR
	country	VARCHAR
	founder	VARCHAR

product

PK/N	product_id	INT
N	name	VARCHAR
N	volume_ml	INT
N	price	DECIMAL
N	intensity	VARCHAR
FK/N/DC/UC	brand_id	INT...

retail

PK/N	retail_id	INT
N	store_name	VARCHAR
FK/N/DC/UC	product_id	INT
	quantity_available	INT
	store_address	VARCHAR
.	region	VARCHAR...

product_note

FK/N/DC/UC	product_id	INT
FK/N/DC/UC	note_id	INT

note

PK/N	note_id	INT
	top	VARCHAR
	heart	VARCHAR
	base	VARCHAR

The Perfume Database schema ensures efficient organization, flexibility, and normalization, supporting detailed management of brands, products, fragrance notes, and retail availability while minimizing redundancy and ensuring data integrity. The **brand** table uniquely identifies each brand with a primary key (brand_id) and includes essential attributes like name, year of establishment, and country. The **product** table connects to the brand table via a foreign key, storing attributes like product name, volume, and intensity, ensuring a one-to-many relationship between brands and their products. The **note** table categorizes fragrance components (top, heart, and base notes) with a unique note_id, enabling detailed fragrance analysis. The **product_note** table establishes a many-to-many relationship between products and notes, ensuring no duplicate pairings. Finally, the **retail** table tracks product availability across stores with attributes like store_name and quantity_available. The schema's constraints, primary keys, foreign keys, not null, on update cascade, on delete cascade, and normalization ensure data consistency, integrity, and efficient operations.