Capsule Skincare Rack 101

Here is my project 1 proposal
Project Title: Capsule Skincare Rack 101
Objective: To help people controlling their impulse of purchase by encouraging users to add whatever skin care product they already own and do more research on whatever product they are interested in purchasing.

Scope:

- Users would be able to add whatever products they have right now
- For each skincare product, users can add reviews and ratings. The reviews can be public or private
- Users can create wishlist that contains the products they would like to purchase

Example Queries:

- Get the top 5 lotions that are rated the best for dry skins
- Get the total price of all the serums in someone's wishlist
- Get the total size of cream someone currently owns

Task 1. Problem Statement

(10 pts) Describe the requirements of the problem with a simple document that lists the rules of the database in the problem domain language. Then describe a list of business rules and the list of possible nouns and actions you identified. I'm expecting this to be a short 1 or 2 pages document.

In today's beauty and skincare industry, it's become a common trend for people to accumulate an overwhelming array of products in pursuit of the perfect skincare routine. With countless options available, from serums and creams to masks and cleansers, consumers often fall into the trap of over-purchasing without considering whether these products are truly suited to their unique skin needs. This impulsive buying not only strains the wallet but can also lead to disappointment as mismatched products may not deliver the desired results. A more informed and minimalist approach, guided by understanding one's skin type and concerns, can be a more effective and cost-efficient way to achieve healthy, glowing skin while reducing clutter in the bathroom cabinet.

Therefore, I would like to create an application that help users to record and manage what skincare products they already own, so that they would always be aware of what they already have before purchasing more. Moreover, I would love to build a platform for users to communicate on their feedback about these products. With the platform, users who are interested in purchasing the products would make a more informed decision.

A list of nouns may include:

- Product
    - Name
    - Price
    - Type
    - Weight

- User
    - Skin type
    - Age
- Wishlist
    - Items that the user would like to purchase
- Feedback
    - Rating
    - Review

A list of actions may include:

- A user can create a wishlist
- A user owns a skincare rack that contains skincare products
- A user can add or remove products to or from the rack
- A user can add or remove products to or from the wishlist
- A user can write feedback about a product
- A user can search for a product and get a collection of feedback

A list of rules may include:

- A user must have one and only one skincare rack
- The skincare rack may not have any skincare product
- A product may have zero feedback
- A product can be categorized into cleanser, lotion, serum, cream, sunscreen, mask, etc.
- Every wishlist has a owner
- A user may have zero wishlist or multiple wishlists
- All prices should be positive if not null
- All size should be non-negative if not null
- One user may write at most one feedback about one product
- All items in the wishlist must exist in the product lists

Task 2. UML

(15 pts) Analyze the problem and create a conceptual model in UML using a tool of your choice (e.g., LucidChart, Enterprise Architect, ArgoUML, Visual Paradigm, ERwin, TOAD) as discussed during class and provided in the references and resources below. Additional requirements and clarifications will be provided in the #general channel on Slack. The diagram must contain at least three classes, at least one to many relationship and one many to many. All relationships, except generalization, must have full multiplicity constraints and labeled as appropriate. Classes must have proper names, descriptions, and attributes with domain types. Key attributes and derived attributes must be marked. Don't build a model with more than 10 entities.

Based on the analysis on what have been described in task 1, the UML diagram may look like below.

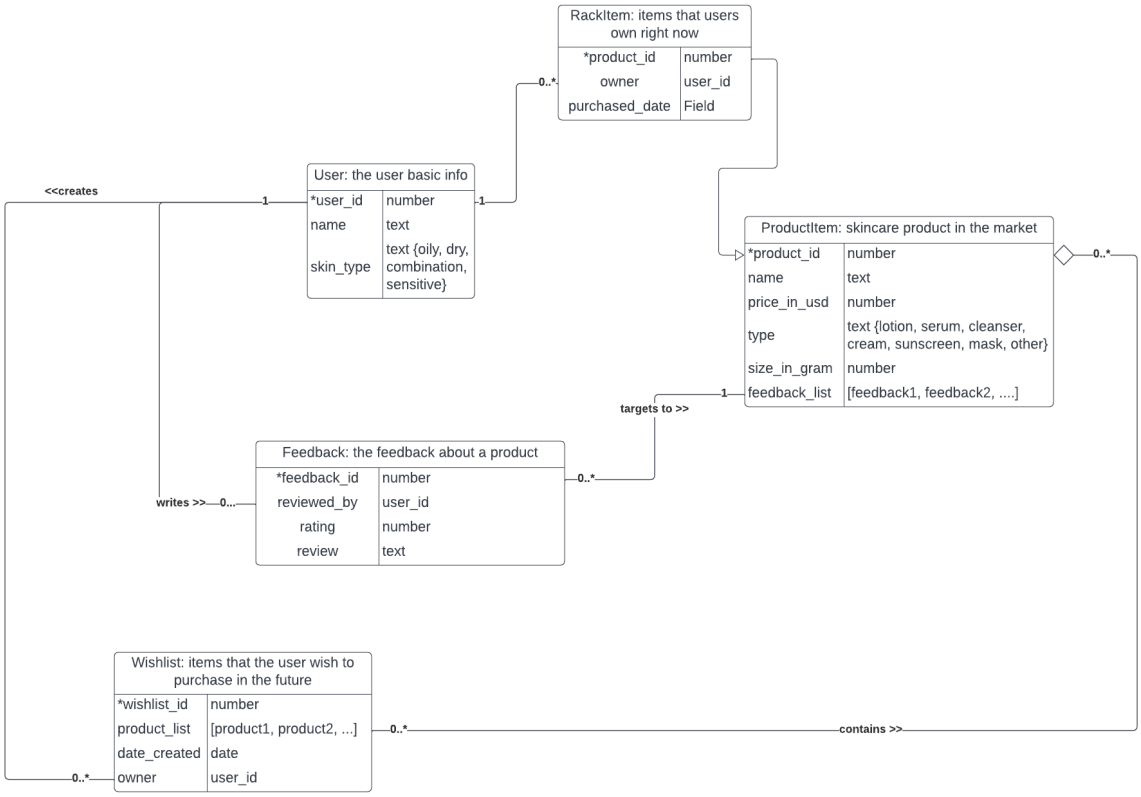


Figure 1. UML diagram

This diagram contains five classes with at least user-to-rank items as one to many relationship and product-to-feedback as many-to-many relationship. I have labeled the multiplicity with the action verb along the lines. I have also labeled the constraints such as skin type is chosen from a given list of skin types. For each entity, I have the names with a short description, as well as the attributes with the assumed type for now. All attributes are important.

Task 3. ERD

(10 pts) From the Conceptual Model, construct a logical data model expressed as an ERD using a language of your choice (other than UML) and a tool of your choice. The logical data model may not have any many-to-many relationships, so introduce association entities as needed.

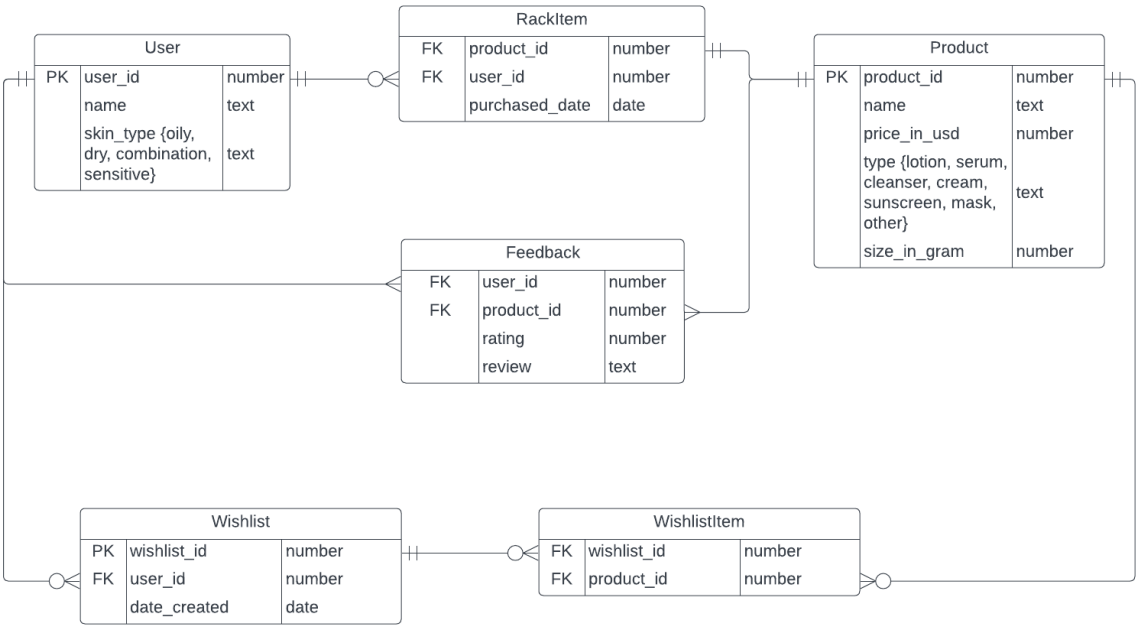


Figure 2. ER Diagram

Task 4. Schema

(15 pts) From the logical model, define a relational schema in at least BCNF. Using functional dependencies, show that the schema in in at least BCNF.

From the ER diagram shown in task 3, the schema for this database is as listed below.
(The primary keys are underlined and foreign keys in *italic* style.)

1. **User Table**
    a. Table name: Users
    b. Attributes: user_id, name, skin_type
    c. There are no non-trivial functional dependencies involving user_id. The Users table remains in BCNF.
2. **Feedback Table**
    a. Table name: Feedbacks
    b. Attributes:*user_id, product_id*, rating, review
    c. The functional dependency "user_id -> name, skin_type" (based on the user_id in the Users table) is non-trivial but involves a superkey (user_id is part of the primary key). The Feedbacks table remains in BCNF.
3. **Product Table**
    a. Table name: Products
    b. Attributes: product_id, price_in_usd, type, size_in_gram

   c. There are no non-trivial functional dependencies involving product_id. The Products table remains in BCNF.
4. **Rack Table**
   a. Table name: Racks
   b. Attributes: *product_id, user_id*, purchased_date
   c. The functional dependency "(product_id, user_id) -> purchased_date" (based on the product_id in the Products table) is non-trivial but involves a superkey. The Racks table remains in BCNF.
5. **Wishlist Table**
   a. Table name: Wishlists
   b. Attributes: wishlist_id, *user_id,* date_created
   c. There are no non-trivial functional dependencies involving wishlist_id. The Wishlists table remains in BCNF.
6. **Wishlist Item Table**
   a. Table name: WishlistItems
   b. Attributes: *wishlist_id, product_id*
   c. Wishlist Items table has the attributes "wishlist_id" and "product_id" and serves solely as a bridge table to establish a many-to-many relationship between Wishlists and Products, with no other attributes or functional dependencies, then it has a valid BCNF schema.

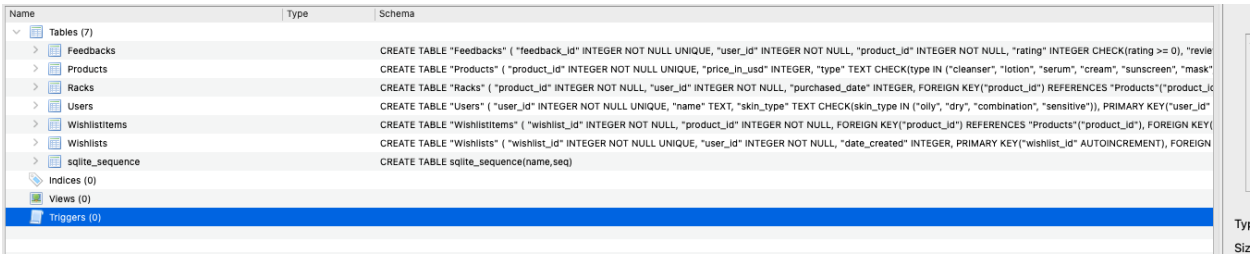
Task 5. SQL data definition statements

(10 pts) Create a set of SQL data definition statements for the above model and realize that schema in SQLite3 by executing the script from the SQLite3, the console or Node. You can use DB Browser to generate these statements. Show that the tables were created and conform to the constraints through screen shots or other means.

Table	SQL
Users	CREATE TABLE "Users" ("user_id" INTEGER NOT NULL UNIQUE, "name"TEXT, "skin_type" TEXT CHECK(skin_type IN ("oily", "dry", "combination", "sensitive")), PRIMARY KEY("user_id" AUTOINCREMENT));
Products	CREATE TABLE "Products" ("product_id" INTEGER NOT NULL UNIQUE, "price_in_usd" INTEGER,

	"type" TEXT CHECK(type IN ("cleanser", "lotion", "serum", "cream", "sunscreen", "mask", "other")), "size_in_gram" INTEGER CHECK(size_in_gram >= 0), PRIMARY KEY("product_id" AUTOINCREMENT));
Feedbacks	CREATE TABLE "Feedbacks" ("user_id" INTEGER NOT NULL, "product_id" INTEGER NOT NULL, "rating"INTEGER CHECK("rating" >= 0), "review" TEXT, PRIMARY KEY("user_id","product_id"), FOREIGN KEY("product_id") REFERENCES "Products"("product_id"), FOREIGN KEY("user_id") REFERENCES "Users"("user_id"));
Racks	CREATE TABLE "Racks" ("product_id" INTEGER NOT NULL, "user_id" INTEGER NOT NULL, "purchased_date" INTEGER, FOREIGN KEY("product_id") REFERENCES "Products"("product_id"), FOREIGN KEY("user_id") REFERENCES "Users"("user_id"));
Wishlists	CREATE TABLE "Wishlists" ("wishlist_id" INTEGER NOT NULL UNIQUE, "user_id" INTEGER NOT NULL, "date_created"INTEGER, PRIMARY KEY("wishlist_id" AUTOINCREMENT), FOREIGN KEY("user_id") REFERENCES "Users"("user_id"));
WishlistItems	CREATE TABLE "WishlistItems" ("wishlist_id" INTEGER NOT NULL, "product_id" INTEGER NOT NULL, PRIMARY KEY("wishlist_id","product_id"), FOREIGN KEY("product_id") REFERENCES "Products"("product_id"), FOREIGN KEY("wishlist_id") REFERENCES "Wishlists"("wishlist_id"));

The screenshot for the tables (taken in db browser):



Task 6. Create test data

(10 pts) Populate the tables with test data. You can use tools such as https://www.mockaroo.com/schemas or https://www.generatedata.com/.

Created test data with mockaroo for all tables. The generated data covers all schemas while respecting the database constraints. For details, please refer to the database file.

Task 7. Queries

(10 pts) Define and execute at least five queries that show your database. At least one query must contain a join of at least three tables, one must contain a subquery, one must be a group by with a having clause, and one must contain a complex search criterion (more than one expression with logical connectors). Experiment with advanced query mechanisms such as RCTE, PARTITION BY, or SELECT CASE/WHEN.

-- List all the wishlist items with the user skin type and the product name (joined 4 tables together)

```
SELECT Users.user_id, Users.name as user_name, Users.skin_type, Products.name as product_name
FROM
WishlistItems
JOIN Wishlists on WishlistItems.wishlist_id = Wishlists.wishlist_id
JOIN Users on Wishlists.user_id = Users.user_id
```

```
JOIN Products on WishlistItems.product_id = Products.product_id
ORDER by Users.user_id
```

-- list all products that has higher than avergage rating (with subqueries)

```
SELECT Products.name as product_name, Products.type, Feedbacks.rating
FROM Products
JOIN Feedbacks on Feedbacks.product_id= Products.product_id
WHERE Feedbacks.rating >
(SELECT  avg(Feedbacks.rating)
from Products
JOIN Feedbacks on Feedbacks.product_id = Products.product_id)
ORDER by Feedbacks.rating
```

-- list the product that has been added by at least 3 users into their wishlist (using group by with having)

```
SELECT Products.name, count(WishlistItems.product_id) as user_wished
 from WishlistItems
 JOIN Products on Products.product_id = WishlistItems.product_id
 GROUP by Products.product_id
 HAVING user_wished > 3
```

-- list the lotion or serum that oily or combination skin has rated in descending order (complex criteria)

```
 SELECT Products.name as product_name, Products.type, Users.skin_type, Feedbacks.rating
from Products
 JOIN Feedbacks on Products.product_id = Feedbacks.product_id
 JOIN Users on Users.user_id = Feedbacks.user_id
 where (Products.type = "lotion" or Products.type = "serum") AND (Users.skin_type = "oily" or
Users.skin_type = "combination")
 ORDER by Feedbacks.rating
```

-- find the best lotion for oily skin

```
 SELECT Products.name as product_name, Products.type, Users.skin_type,
avg(Feedbacks.rating) as avg_rating from Products
 JOIN Feedbacks on Products.product_id = Feedbacks.product_id
 JOIN Users on Users.user_id = Feedbacks.user_id
 where Products.type = "lotion" AND Users.skin_type = "oily"
```

```
 GROUP by Products.product_id
ORDER by avg_rating DESC
LIMIT 1
```

Task 8. Node + Express

(20 pts) Create a basic Node + Express application that let's you create, display, modify and delete at least two of the tables with a foreign key between then. No need to have a polished interface, and you can use the code created in class as a starting point

a. https://github.com/john-guerra/nodeExpressSqliteEJS_2/tree/main
b. https://github.com/john-guerra/nodeExpressSqliteEJS_Bikeshare