**basic user-based collaborative filtering recommendation system** in SQL.

**Goal:**

Recommend items to **User 1** based on what **similar users** liked (and User 1 hasn't rated yet).

**Assumptions:**

You have a table:

sql

CopyEdit

ratings(user\_id INT, item\_id INT, rating FLOAT)

**Step-by-Step Query (Using Cosine Similarity)**

sql

CopyEdit

-- Step 1: Find users who have rated the same items as User 1

WITH common\_items AS (

SELECT

r1.user\_id AS user1,

r2.user\_id AS user2,

r1.item\_id,

r1.rating AS rating1,

r2.rating AS rating2

FROM ratings r1

JOIN ratings r2 ON r1.item\_id = r2.item\_id

WHERE r1.user\_id = 1 AND r2.user\_id != 1

),

-- Step 2: Calculate cosine similarity between User 1 and all other users

similarities AS (

SELECT

user2,

SUM(rating1 \* rating2) AS dot\_product,

SQRT(SUM(POWER(rating1, 2))) AS norm1,

SQRT(SUM(POWER(rating2, 2))) AS norm2,

SUM(rating1 \* rating2) / (SQRT(SUM(POWER(rating1, 2))) \* SQRT(SUM(POWER(rating2, 2)))) AS cosine\_similarity

FROM common\_items

GROUP BY user2

),

-- Step 3: Get items rated by similar users that User 1 hasn't rated yet

unseen\_items AS (

SELECT

r.user\_id,

r.item\_id,

r.rating

FROM ratings r

WHERE r.item\_id NOT IN (

SELECT item\_id FROM ratings WHERE user\_id = 1

)

),

-- Step 4: Predict score for unseen items based on similarity-weighted ratings

recommendations AS (

SELECT

u.item\_id,

SUM(s.cosine\_similarity \* u.rating) / SUM(s.cosine\_similarity) AS predicted\_score

FROM unseen\_items u

JOIN similarities s ON u.user\_id = s.user2

GROUP BY u.item\_id

)

-- Final output: Top recommended items for User 1

SELECT \* FROM recommendations

ORDER BY predicted\_score DESC

LIMIT 5;

**What This Does:**

* **Finds similar users** to User 1 using cosine similarity.
* **Finds items** those users have rated that User 1 hasn’t.
* **Predicts a score** for each of those unseen items using similarity-weighted average.
* **Recommends the top items**.

If I say I used collaborative filtering in grocery system in a way that once the user add his dislike ingredients the system filters through all the items user want and would not recommend a item because of this dislike ingredients is present in item

For such application to work it is **not exactly collaborative filtering**, but more of a **content-based filtering** or a **rule-based filtering system**.

Let me break it down clearly:

**What You Did (based on our description):**

* User adds ingredients they dislike.
* When they search or want recommendations, items (like grocery products or recipes) that contain those ingredients are **filtered out**.
* This is based on the **content of the item** (ingredients), **not other users’ behavior**.

**Why It’s Not Collaborative Filtering**

Collaborative filtering relies on:

* Similar users (user-user CF)
* Similar items based on user behavior (item-item CF)
* It **doesn’t care what the item contains**, only how users interact with it.

**What we Actually Did:**

we implemented a **content-based filtering system with user-defined preferences** (dislikes). This is totally valid and very useful, especially in food/grocery systems.

we could say something like:

"I used a content-based filtering system in the grocery platform, where user-defined preferences (such as disliked ingredients) are used to filter out recommendations. While collaborative filtering suggests items based on user behavior patterns, my system focuses on the actual item content, ensuring users don’t see products containing ingredients they want to avoid."

**How You Can Combine Both (if you want to sound smart):**

You can say you plan to combine **collaborative filtering** and **content-based filtering** into a **hybrid recommendation system**. That’s how many modern systems work (like Netflix, Amazon, etc.).

For that ! Here's a clean and professional explanation

Implemented a content-based filtering system in a grocery recommendation platform that personalizes results by excluding items containing user-defined disliked ingredients.  
Explored combining this with collaborative filtering to enhance recommendations by integrating both user preferences and behavior patterns, aiming for a hybrid approach similar to real-world systems like Amazon or Netflix.

Sure! Here’s a simple **Python implementation** of the logic you described: filtering grocery items based on **disliked ingredients**.

We’ll simulate:

* A list of **grocery items**, each with a set of ingredients.
* A **user preference** with disliked ingredients.
* Filtering out items that contain any disliked ingredient.

**Python Code Example:**

python

CopyEdit

# Sample grocery items with their ingredients

grocery\_items = [

{"name": "Chicken Curry", "ingredients": {"chicken", "onion", "garlic", "spices"}},

{"name": "Veg Salad", "ingredients": {"lettuce", "tomato", "onion", "cucumber"}},

{"name": "Fruit Yogurt", "ingredients": {"yogurt", "strawberry", "sugar"}},

{"name": "Pasta Alfredo", "ingredients": {"pasta", "cheese", "milk", "garlic"}},

]

# User-defined disliked ingredients

user\_dislikes = {"onion", "garlic"}

# Filter function

def filter\_items(items, dislikes):

return [

item for item in items

if not item["ingredients"] & dislikes # intersection means it has a disliked ingredient

]

# Get filtered recommendations

filtered\_recommendations = filter\_items(grocery\_items, user\_dislikes)

# Output

print("Recommended items (excluding disliked ingredients):")

for item in filtered\_recommendations:

print("-", item["name"])

**Output:**

diff

CopyEdit

Recommended items (excluding disliked ingredients):

- Fruit Yogurt

This is a basic **content-based filtering logic**. If you want to extend this by combining it with collaborative filtering or connect it to a real database,

* **Collaborative Filtering (user similarity)** – we'll simulate user ratings.
* **Content-Based Filtering** – we’ll still filter out items with disliked ingredients.

**Scenario Setup:**

* Each user rates grocery items (like 1 to 5).
* User 1 has disliked ingredients they want to avoid.
* We'll:
  1. Find similar users to User 1 using **cosine similarity**.
  2. Predict items User 1 might like based on similar users’ ratings.
  3. Filter out items that contain User 1’s disliked ingredients.

**Python Implementation (Hybrid Recommender)**

python

CopyEdit

from math import sqrt

from numpy import dot

from numpy.linalg import norm

# --- Data ---

# User ratings: user\_id -> {item\_name: rating}

user\_ratings = {

1: {"Chicken Curry": 5, "Veg Salad": 2},

2: {"Chicken Curry": 4, "Pasta Alfredo": 5, "Fruit Yogurt": 3},

3: {"Veg Salad": 4, "Fruit Yogurt": 5, "Pasta Alfredo": 2}

}

# Grocery items with ingredients

grocery\_items = {

"Chicken Curry": {"chicken", "onion", "garlic", "spices"},

"Veg Salad": {"lettuce", "tomato", "onion", "cucumber"},

"Fruit Yogurt": {"yogurt", "strawberry", "sugar"},

"Pasta Alfredo": {"pasta", "cheese", "milk", "garlic"}

}

# User 1's dislikes

user\_dislikes = {"onion", "garlic"}

# --- Cosine similarity function ---

def cosine\_similarity(u1, u2):

items = set(u1.keys()) & set(u2.keys())

if not items:

return 0

v1 = [u1[i] for i in items]

v2 = [u2[i] for i in items]

return dot(v1, v2) / (norm(v1) \* norm(v2))

# --- Step 1: Find similarity scores ---

target\_user = 1

similarities = {}

for other\_user, ratings in user\_ratings.items():

if other\_user != target\_user:

sim = cosine\_similarity(user\_ratings[target\_user], ratings)

similarities[other\_user] = sim

# --- Step 2: Predict ratings for unseen items ---

predicted\_ratings = {}

for item in grocery\_items:

if item not in user\_ratings[target\_user]:

numerator = 0

denominator = 0

for other\_user, sim in similarities.items():

if item in user\_ratings[other\_user]:

numerator += sim \* user\_ratings[other\_user][item]

denominator += sim

if denominator > 0:

predicted\_ratings[item] = numerator / denominator

# --- Step 3: Filter out items with disliked ingredients ---

final\_recommendations = {

item: score for item, score in predicted\_ratings.items()

if not grocery\_items[item] & user\_dislikes

}

# --- Output ---

print("Hybrid Recommendations for User 1:")

for item, score in sorted(final\_recommendations.items(), key=lambda x: x[1], reverse=True):

print(f"- {item}: Predicted Rating {score:.2f}")

**What This Does:**

* Calculates **similarity** between User 1 and others.
* Predicts ratings for **unseen items** using collaborative filtering.
* **Filters out** items with ingredients User 1 dislikes.
* Recommends the **best remaining items**.