University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 4

Date: November 28th, 2025

Group Number: 39

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Sarah Liang	45379450	j8i7t	sarahliang13013@gmail.com
Satsuki Onome	50010305	d1k9c	sonome2005@gmail.com
Dayna Yoon	51355618	s9g8l	dawon020411@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

University of British Columbia, Vancouver
Department of Computer Science

Project Summary:

The domain of this application is culinary and nutrition management, and more specifically, is focused around recipe organization. It encompasses areas such as storing recipes, giving instructions for cooking, and tracking nutritional or dietary information. Moreover, the system also includes community engagement features such as rating or saving recipes, which extend the domain into social and collaborative cooking.

Edits in the recipe_schema.sql:

- added recipeID attribute (FK references Recipe(ID)) to SavedLists entity. Due to this modification, the order of creating the tables and inserting values was modified to avoid error.
- More tuples were added so that all functionalities of the queries can be tested against.

University of British Columbia, Vancouver
Department of Computer Science

Final Schema Diagram (Screenshot)

1	**Customer**				
2	**ID**	**name**	**email_address**		
3	1	Satsuki Onome	satsuki@example.com		
4	2	Dayna Yoon	dayna@example.com		
5	3	Sarah Liang	sarah@example.com		
6	4	cpsc304	cpsc304@example.com		
7	5	ubc cs	ubccs@example.com		
8	100	food critic	food_critic@example.com		
9	200	recipe creator	recipe_creator@example.com		
10	300	just customer	just_customer@example.com		
11					
12	**Cuisine**				
13	**ID**	**style**			
14	1	Japanese			
15	2	Italian			
16	3	Korean			
17	4	Chinese			
18	5	Western			
19					
20	**Recipe**				
21	**ID**	**title**	**time_consumed**	**difficulty**	**cuisineID**
22	1	Pasta Carbonara	25	Medium	2
23	2	Tteokbokki	20	Easy	3
24	3	Sushi	45	Hard	1
25	4	Fried Rice	15	Easy	4
26	5	Burger	30	Medium	5
27	6	Pasta Bolognese	25	Medium	2
28	7	Fried Chicken	40	Medium	5
29	8	Pizza	45	Hard	2
30	9	Kimbap	30	Medium	3
31	10	Dimsum	60	Hard	4
32					

33	**SavedLists**			
34	**ID**	**name**	**ownerID**	**recipeID**
35	1	Favorites	1	1
36	2	To Try	2	2
37	12	Favorites	2	5
38	3	High Protein	3	3
39	13	High Protein	3	4
40	4	Spicy Food	4	1
41	5	Comfort Meals	5	2
42				
43	**Instruction**			
44	**step**	**recipeID**	**notes**	
45	1	1	Boil pasta until al dente.	
46	2	1	Mix egg and cheese in bowl.	
47	1	2	Boil rice cakes in spicy sauce.	
48	1	3	Prepare sushi rice and roll with fillings.	
49	1	4	Stir-fry rice with vegetables.	
50				
51	**Ingredient**			
52	**ID**	**name**		
53	1	Egg		
54	2	Peanut		
55	3	Pasta Noodle		
56	4	Cheese		
57	5	Soy		
58	6	Tomato		
59	7	Lettuce		
60	8	Garlic		
61	9	Milk		
62	10	Chicken		

64	**Allergen**		
65	**ID**	**name**	
66	1	Egg	
67	2	Dairy	
68	3	Gluten	
69	4	Peanut	
70	5	Soy	
71			
72	**Nutrition 1**		
73	**name**	**grams**	**recipeID**
74	Protein	12.5	1
75	Fat	8.3	1
76	Carbs	65.2	2
77	Protein	10	3
78	Fat	5.4	4
79			
80	**Nutrition 2**		
81	**ID**	**name**	**recipeID**
82	1	Protein	1
83	2	Fat	1
84	3	Carbs	2
85	4	Protein	3
86	5	Fat	4
87			
88	**Fresh**		
89	**ID**	**location**	
90	2	Vancouver Farm Market	
91	5	Vancouver Farm Market	
92	6	BC Tomato Farm	
93	7	Green Leaf Farm	
94	10	Free Range Poultry	

96	**Processed**		
97	**ID**	**companyName**	
98	1	Golden Eggs Farm.	
99	3	Walmart	
100	4	Dairyland Foods	
101	8	Garlic Picks.	
102	9	Happy Cow Dairy	
103			
104	**RecipeCreator**		
105	**ID**	**cookingHistory**	
106	1	Created 10 recipes this month	
107	2	Specializes in Korean dishes	
108	3	Focuses on Italian fusion	
109	4	Tests community recipes	
110	5	New creator	learning sushi
111	200	recipe professional	
112			
113	**FoodCritic**		
114	**ID**	**ratingHistory**	
115	1	Rated 50 recipes in total	
116	2	Prefers spicy foods	
117	3	Writes detailed reviews	
118	4	Cares about presentation	
119	5	Values ingredient quality	
120	100	food criticizing professional	

122	**AddRelation**	
123	**CustomerID**	**RecipeID**
124	1	1
125	1	2
126	2	3
127	3	4
128	4	5
129	1	6
130	3	7
131	4	8
132	2	9
133	3	10

178	**Contain**	
179	**RecipeID**	**IngredientID**
180	1	3
181	1	1
182	1	4
183	2	2
184	4	5
185	6	3
186	6	4
187	6	6
188	6	8
189	8	4
190	7	10
191		
192	**RecipeReference**	
193	**RecipeID**	**ReferenceID**
194	1	2
195	2	3
196	3	4
197	4	5
198	5	1
199		
200	**CanHave**	
201	**IngredientID**	**AllergenID**
202	1	1
203	4	2
204	3	3
205	5	5
206	2	4

135	Rate		
136	**CustomerID**	**RecipeID**	**stars**
137	1	1	5
138	2	1	5
139	3	1	4
140	4	1	5
141	5	1	5
142	100	1	5
143	200	1	4
144	300	1	5
145	1	2	4
146	2	2	3
147	3	2	5
148	4	2	4
149	5	2	4
150	100	2	3
151	200	2	4
152	300	2	3
153	1	3	2
154	2	3	2
155	3	3	3
156	4	3	1
157	5	3	2
158	100	3	2
159	200	3	1
160	300	3	2
161	1	4	4
162	2	4	4
163	3	4	3
164	4	4	5
165	5	4	4
166	100	4	4
167	200	4	5
168	300	4	3
169	1	5	5
170	2	5	5
171	3	5	4
172	4	5	5
173	5	5	5
174	100	5	5
175	200	5	5
176	300	5	4

---

1. **INSERT**

Query: Insert a recipe with customerID, recipeID, title, time_consumed, difficulty, and cuisineID

Table that affects: Recipe, AddRelation

Table that is used: Recipe, AddRelation, Customer (condition checking)

Values: customerID: INTEGER, recipeID: INTEGER, title: CHAR(100), time_consumed: INTEGER, difficulty: CHAR(20), cuisineID: INTEGER

Error handling:

- Unique recipe title. Case-insensitive.
- Unique recipe ID.
- cuisineID and Customer must exist.

sql: appService.js rows: 127-177

```sql
INSERT INTO Recipe (id, title, time_consumed, difficulty, cuisineID)
        VALUES (:id, :title, :time_consumed, :difficulty, :cuisineID)
INSERT INTO AddRelation (CustomerID, RecipeID)
        VALUES (:customerID, :id)
```

2. **UPDATE**

Query: UPDATE a Customer's name and/or email_address by selecting an existing Customer.

relation: `Customer`

updated attributes:

name (non-PK), email_address (non-PK, UNIQUE)

PK: ID

CK: email_address

sql: appService.js rows: 396-425

sql:

```sql
UPDATE Customer
SET name = :newName,
   email_address = :newEmail
WHERE ID = :customerID;
```

3. **DELETE**

Query: Delete an ingredient by ID. Deleting the ingredient also removes related tuples in Fresh, Processed, Contain, and CanHave via ON DELETE CASCADE.

sql:

```sql
DELETE FROM Ingredient
WHERE ID = :id;
```

sql: appService.js rows: 320-340

4. **SELECTION**

Query: SELECT FoodCritic or RecipeCreator and/or name. Show all customer information that matches those conditions.(RecipeCreator has cookingHistory and FoodCritic has ratingHistory information shown as well)

Relation used: Customer(Parent relation), RecipeCreator, FoodCritic

sql: appService.js rows: 198-234

User chooses AND / OR via dropdown → passed in as :andOr:

```sql
SELECT C.ID, C.name, C.email_address, RC.cookingHistory AS history
FROM Customer C
JOIN RecipeCreator RC ON C.ID = RC.ID
JOIN FoodCritic FC ON C.ID = FC.ID
ORDER BY C.ID
```

with name:

```sql
SELECT C.ID, C.name, C.email_address, RC.cookingHistory, FC.ratingHistory
  FROM Customer C
  LEFT JOIN RecipeCreator RC ON C.ID = RC.ID
  LEFT JOIN FoodCritic FC ON C.ID = FC.ID
  WHERE C.name = :name
  ORDER BY C.ID,
```

5. **PROJECTION**

Query: Show only the selected attributes of the Recipe relation based on user selection.

Relation: Recipe

---

attributes:

ID: INTEGER

title: CHAR

time_consumed: INTEGER

difficulty: CHAR

cuisineID: INTEGER

User input: User selects (checkbox) which attributes to display (any subset)

sql: appService.js rows: 429-443

sql:

```sql
SELECT <user-selected attributes>
FROM Recipe
```

6. **JOIN**

Query: For a given recipe title and minimum rating, find all customers who rated that recipe with at least that many stars, showing customer info and their rating.

sql: appService.js rows: 342-365

sql:

```sql
SELECT C.ID, C.name, C.email_address, R.stars
FROM Customer C
JOIN Rate   R  ON C.ID = R.CustomerID
JOIN Recipe Re ON R.RecipeID = Re.ID
WHERE LOWER(TRIM(Re.title)) LIKE '%' || LOWER(TRIM(:recipeTitle)) || '%'
```

AND R.stars >= :minStars

ORDER BY C.ID

```

### 7. AGGREGATION with GROUPBY

Query: Find and show the number of recipes in each savedList of the owner, alongside with the savedList ID and name.

Aggregation used: COUNT

**sql: appService.js rows: 236-252**

```sql
SELECT
    S.name AS savedListName,
    C.name AS ownerName,
    COUNT(S.recipeID) AS recipeCount
FROM SavedLists S
JOIN Customer C ON S.ownerID = C.ID
GROUP BY S.name, C.name, S.ownerID
ORDER BY C.name, S.name;
```

8. AGGREGATION with HAVING

Query: Find recipe titles whose average rating is greater than or equal to a user-selected threshold.

Relations used: Recipe, Rate

User input: threshold (e.g., 4.0, 4.5)
```

---

sql: appService.js rows: 446-468

sql:

```sql
SELECT R.title, AVG(RT.stars) AS avg_rating
FROM Recipe R
JOIN Rate RT ON R.ID = RT.RecipeID
GROUP BY R.ID, R.title
HAVING AVG(RT.stars) >= :threshold
ORDER BY avg_rating DESC;
```

9. **NESTED AGGREGATION with GROUPBY**

Query: Find the cuisine style(s) with the highest average recipe rating.

sql: appService.js rows: 368-394

sql:

```sql
SELECT Cu.style, AVG(R.stars) AS avgCuisineRating
FROM Cuisine Cu
JOIN Recipe Re ON Cu.ID = Re.cuisineID
JOIN Rate R   ON R.RecipeID = Re.ID
GROUP BY Cu.style
HAVING AVG(R.stars) >= ALL (
        SELECT AVG(R2.stars)
        FROM Cuisine Cu2
```

```
        JOIN Recipe Re2 ON Cu2.ID = Re2.cuisineID

        JOIN Rate R2   ON R2.RecipeID = Re2.ID

        GROUP BY Cu2.style)
```

10. **DIVISION**

Query: Find all recipes that contain all ingredients selected by the user. When no ingredients are selected, it shows all recipes. (up to 5 ingredients can be chosen) Case-insensitive.

relations used: Ingredients, Contains, Recipe

sql: appService.js rows: 254-303

```sql
SELECT R.ID, R.title

FROM Recipe R

JOIN Contain C ON R.ID = C.RecipeID

JOIN Ingredient I ON C.IngredientID = I.ID

WHERE LOWER(TRIM(I.name)) IN (:ing0, :ing1, :ing2, :ing3, :ing4)

GROUP BY R.ID, R.title

HAVING COUNT(DISTINCT LOWER(TRIM(I.name))) = :ingCount
```