Group 25
'DROP TABLE project;
Lauren Baldino, Sierra Dacey, Alanna Luce, Christine O'Brien

Recipe Domain Project

Introduction

We will develop an application in Python to provide a command-line interface to a recipe database. The application will be simultaneously accessible by multiple users. The database will store Recipe, Ingredient and User data. Source data scraped from websites via Python script and parsed from library/ebook sources will be normalized using regular expressions. Units will be standardized before data is imported into the database, so that each ingredient is only measured in one unit.

Domain

The overall objective of our domain is to provide a system to let users search for and share recipes and track their available cooking ingredients.

EER Modeling

Ingredients
Ingredients contain attributes available to view such as item name, expiration date, quantity, unit, aisle location, and purchase date. Users and ingredients have a 0-to-N relationship because a user can use as many ingredients as are available in their pantry to complete a recipe. One or more ingredients can be used to make recipes. Each ingredient is linked to a unit when added and all subsequent recipes and pantry entries will be measured in that unit.

Pantry
We did not treat the pantry as an entity in the EER diagram because all information in the Pantry specification reflects the M-to-N user-to-ingredient relationship. Each user-to-ingredient association has attributes of quantity bought, current quantity, aisle, purchase date, and expiration date.

Users
Users have accounts with a unique username and a password that give them access to the database. The user account contains the creation date of the account, and the last date they last accessed the database. Users have two separate relationships to Recipes; neither are mandatory. Users are able to contribute any number of recipes to the database, and can make any number of recipes from it. In addition, users can track their cooking ingredients to determine which recipes they can make.

Recipes
Recipes have attributes such as name, description, servings, cook time, difficulty, rating, categories, ingredients, and steps. Each recipe has a unique id so that different recipes that have the same name can be stored. Each recipe has one or more ingredients, creating a 1-to-N relationship. Recipes can be linked to one or more categories but are not required to be in any. Each Recipe has one user as contributor and has a creation date indicating when it was added to the database. Recipes can be linked to an optional Original Source tracking the site or publication from which the recipe came from.

Original Source
Original Source is an optional attribution of a recipe to a source publication such as a book or website. A recipe can have 0 or 1 original sources, and a source contributes 1-to-N recipes. This was added to the original specification to provide dimensions for meaningful data analysis and to credit sources. Each source should have a unique combination of publication date and name, to disambiguate volumes of serialized content (magazines, newspapers) and different editions of books.

Reduction to Tables

Ingredients
Each ingredient is associated with exactly one unit so the Ingredient table links directly to the Unit table via a foreign key.

Pantry
Because this is a many-to-many relationship, Pantry does become a table/relation in the Reduction to Tables/relational model. The relationship information is stored here along with the primary keys for User and Ingredient.

Users and Makes
Since many users can make the same recipe, the Makes table stores this M-to-N relationship along with the date the recipe was made and the user rating of it.

Recipes
Since each recipe has one user as contributor and one creation date, this relationship is stored directly in the Recipes table with a foreign key linking to the primary key of Users. The source is identified by a single integer id; the foreign key is included in Recipes because the column would not create a lot of overhead even with potential null values in user-created data. Recipes are linked to the multi-valued Category attribute via the Categorization table.

Difficulty
Since the recipe difficulty will use repeated values such as ‘easy’ and ‘hard,’ the text for these choices will be stored in a separate Difficulty table and the Recipe table will be linked via foreign key.

Categories
Category is an optional multi-valued attribute that can be applied to a recipe. The many-to-many relationship is stored in the Categorization table. The category name is stored in a Category table to allow simultaneous modification of all instances of a category name.

Units
This table allows for efficient storage of the varchar unit name values and provides the ability to update all instances of a unit name simultaneously.

Original Source
Since Original Source and Recipe have a one-to-many relationship with many recipes sharing a source, the Sourced_from table will store these relationships.

Implementation

Design Changes
The user information table was defined as users, rather than user as initially diagrammed, to avoid the SQL reserved word user. We also added the 'aisle' table because it was apparent that this attribute would use a set of repeated values.

Data Manipulation and Loading
Our demo data set was pulled from HelloFresh because that site provided values for all the attributes of our recipe table; many other data sets or online sources were missing one or more attributes. We parsed the many-to-many relationships for recipe categories and recipe ingredients into separate delimited text files, linked to the recipes by a temporary ID, before we added anything to SQL. This site (and most other sites and data sets) stored recipe ingredients as a list of strings, each containing a quantity, a unit, and an ingredient name. For example, a recipe might list [2 cups flour, 3 tablespoons butter,…]. We used regular expressions in Notepad++ to split this into a delimited file with quantity, unit, and name. Even with data from a single source, some ingredients were measured in more than one quantity, so we converted these to consistent units in a temporary database.

Once data was split into delimited files with the proper cardinality and fields, we loaded temporary tables into the PostgreSQL database using DataGrip. We first populated the aisle, category, difficulty, unit, source, and users, since these didn't refer to any other tables. We were then able to populate ingredients (referencing unit) and recipe (referencing difficulty, source, and users). Finally, we were able to populate pantry, recipe_categorization, recipe_ingredients, and recipe_made, which referenced recipe and ingredient.

Since the default search/comparison behavior in our database is case sensitive, we converted all ingredient names to lowercase to avoid handling this separately every time we queried that column.

Sample Queries

Data Definition Language - Table Creation

```
create table users(
    username varchar(100) not null primary key,
    last_login timestamp,
    creation_date timestamp not null,
    password varchar(100) not null
    );

create table ingredient(
    id int not null primary key generated by default as identity,
    name varchar(500) not null unique,
    unit_id int not null,
    foreign key (unit_id) references unit(id)
    );

create table recipe(
    id int not null primary key generated by default as identity,
    name varchar(500) not null,
    description varchar,
    servings int,
    difficulty_id int not null default 3,
    cook_time int,
    steps varchar,
    username varchar(100),
    creation_date timestamp,
    source_id int,
    foreign key (username) references users(username),
    foreign key (difficulty_id) references difficulty(id),
    foreign key (source_id) references source(source_id)
    );

create table pantry(
    username varchar(100),
    ingredient_id int,
    expiration_date date,
    quantity_purchased int,
    current_quantity int,
    purchase_date date,
    primary key(username, ingredient_id),
    foreign key (username) references users(username),
    foreign key (ingredient_id) references ingredient(id)
```

```
    );

create table recipe_ingredients(
    recipe_id int not null,
    ingredient_id int not null,
    quantity numeric,
    foreign key (recipe_id) references recipe(id),
    foreign key(ingredient_id) references ingredient(id),
    primary key(recipe_id, ingredient_id)
    );
```

Data Manipulation - Bulk Loading

```
-- users: add an initial username
insert into users (username, last_login, creation_date, password) values
    ('co3327@rit.edu', current_timestamp, current_timestamp, 'password1');

-- users: add bulk data from temp table
insert into users
    (username,
    last_login,
    creation_date,
    password)
select
    (Username,
    Password,
    Creation_dateTime,
    last_login_dateTime)
from _random_users_dt;

-- Add difficulties to difficulty
insert into difficulty (name) values
    ('easy'),
    ('easy-medium'),
    ('medium'),
    ('medium-hard'),
    ('hard');

-- Copy ingredient info from temp table
insert into ingredients
    (name,
    unit_id)
select
    t.name,
    u.id
from _ingredients t
```

```
inner join units u on t.unit = u.name

-- Copy recipe information from a temp table
insert into recipe
   (id,
   source_id,
   creation_date,
   username,
   steps,
   cook_time,
   difficulty_id,
   servings,
   description,
   name)
select
   t.temp_id,
   s.source_id,
   current_timestamp,
   'co3327@rit.edu',
   t.instructions,
   t.cooktime,
   d.id,
   t.servings,
   t.description,
   t.title
from _temp_recipes t
inner join difficulty d on t.difficulty = d.name
inner join sources s on t.source_name = s.source_title
   and t.source_date = s.publication_date;
```

Data Manipulation - Program Execution

```
-- Account creation
insert into users
   (username,
   password,
   last_login,
   creation_date)
values
   ('John',
   'asdf',
   current_timestamp,
   current_timestamp);

-- Updating last login date
update users set last_login = current_timestamp
```

```
where username = 'John';

-- Adding a new category and returning its id
insert into category (name)
values ('breakfast food') returning id;

-- Adding a recipe categorization
insert into recipe_categorization (recipe_id, category_id)
values (100, 5);

-- Adding to the recipe table
insert into recipe
   (creation_date,
   username,
   name,
   cook_time,
   difficulty_id,
   servings,
   description,
   steps)
values
   (current_timestamp,
   'John',
   '7-UP CHEESE ASPIC',
   20,
   2,
   6,
   'Seven-Up can be used in any cheese salad. You'll have guests asking you for this special recipe.',
   'Dissolve gelatin in hot water. Add chilled 7-Up and onion. Chill until slightly thickened. Fold in remaining ingredients. Turn into one-quart mold; chill until firm. Unmold. Serve on crisp lettuce. Garnish with tomato wedges and cheese cubes. 6 servings.')
   returning id;

-- Adding recipe_ingredients
insert into recipe_ingredients
   (recipe_id,
   ingredient_id,
   quantity)
values
   (100,
   1,
   6);
```

Analysis

Additional recipe and user interaction data from a Kaggle dataset was loaded into the PostgreSQL database using DataGrip. SQL queries, listed in the appendix, were performed on this data to retrieve information about commonly paired ingredients, top recommended recipes, and user retention. Microsoft Excel was used to visualize data trends, including graphing the relationship between the frequency of a user making and rating recipes and the total number of interactions they have with the application.

Indexing and Performance

Since ingredients and recipes are sometimes searched by name, both recipe and ingredient have an index on their name attribute. A username index on recipe supports the 'My Recipes' feature, and an index on creation_date supports the 'most recent recipes' recommender. To support the recommender based on pantry availability and the lookups used during mark-as-made, there's an index in pantry on username and ingredient_id.

Appendix - Queries Used in Phase 4

Addition of rating to marking a recipe as made:

```
insert into recipe_made
   (date_made,
   username,
   recipe_id,
   rating)
values
   (current_timestamp,
   'uname',
   recipe_id,
   rating);
```

Sort search by best rated:

```
select
   name,
   weighted_rating
from recipe inner join
   (select
      recipe_id AS recipe_id,
      sum(rating)/ count(*) as weighted_rating
   from recipe_made
   group by recipe_id)
as rating_info on recipe.id = rating_info.recipe_id
order by weighted_rating desc
```

Adding a new ingredient into the database:

```
-- Get the unit for the ingredient:
      select id
      from unit
      where name = 'unit_name';

-- If the unit does not yet exist:
      insert into unit (name)
      values('unit_name')
      returning id;

--Insert the ingredient
   insert into ingredient
      (name,
```

```
    unit_id)
  values
    ('ingredient_name',
    unit_id)
  returning id;
```

Get a recipe by its id:

```
select
  r.id,
  r.name,
  r.description,
  r.servings,
  d.name,
  r.cook_time,
  r.steps
from recipe r
inner join difficulty d on r.difficulty_id = d.id
where r.id = recipe_id;
```

Get top 50 rated recipes:

```
select
  r.id,
  r.name,
  r.creation_date,
  r.username,
  avg(rm.rating)
from recipe r
inner join recipe_made rm on r.id = rm.recipe_id
where rm.rating is not null
group by r.id
order by avg(rm.rating) desc limit 50;
```

Get newest 50 recipes:

```
select
  id,
  name,
  creation_date,
  username
from recipe
order by creation_date desc limit 50;
```

Get recipes that can be made using current pantry ingredients:

```
select
   r.id,
   r.name,
   r.creation_date,
   r.username,
   avg(rm.rating)
from recipe r
inner join recipe_ingredients ri on r.id = ri.recipe_id
inner join pantry p on ri.ingredient_id = p.ingredient_id
left join  recipe_made rm on r.id = rm.recipe_id
where p.username = 'uname'
and ri.quantity < COALESCE(p.current_quantity, p.quantity_purchased)
and r.id not in
   (select r2.recipe_id
   from recipe_ingredients r2
   where r2.ingredient_id not in
      (select distinct p2.ingredient_id
      from pantry p2
      where p2.username = 'uname' ))
group by r.id
order by avg(rm.rating) desc nulls last;
```

Get recipes made by similar users:

```
select
   r.id,
   r.name,
   r.creation_date,
   r.username,
   avg(rm.rating)
from recipe r
inner join recipe_made rm on r.id = rm.recipe_id
where rm.username in
   (select distinct rm2.username
   from recipe_made rm1
   inner join recipe_made rm2 on rm1.recipe_id = rm2.recipe_id
   where rm1.username = 'uname')
and r.id not in
   (select recipe_id
```

```
    from recipe_made
    where username = ‘uname’)
group by r.id
order by avg(rm.rating) desc;
```

Bulk loading additional data from Kaggle dataset:

```
 -- users: add bulk data from temp tables
insert into users
    (username,
    last_login,
    creation_date,
    password)
select distinct
    (user_created,
    'pwd',
    '2013-1-1',
    current_timestamp)
from _kaggle_temp_recipes;

insert into users
    (username,
    last_login,
    creation_date,
    password)
select distinct
    (reviewer,
    'pwd',
    '2013-1-1',
    current_timestamp)
from _kaggle_temp_reviews
where reviewer not in (select distinct username from users);

-- Copy recipe information from a temp table
insert into recipe
    (source_id,
    creation_date,
    username,
    steps,
    cook_time,
    servings,
    description,
    name)
select
    4,
    date_published,
    user_created,
```

```
    steps,
    total_time_int,
    servings_int,
    description,
    name
from _kaggle_temp_recipes;

-- Copy ingredient info from temp table
-- add any new ingredients; kaggle dataset lacked unit information; using a default here
insert into ingredient
    (name,
    unit_id)
select distinct
    ingredient
from _temp_kaggle_ingredients
where ingredient not in
(select distinct name from ingredient);

-- add the recipe-ingredient information
insert into recipe_ingredients (recipe_id, ingredient_id, quantity)
select r.id, i.id, ti.qty_numeric
from _temp_kaggle_ingredients ti
inner join _kaggle_temp_recipes tr on t.recipetempid = tr.id
inner join recipe r on tr.name = t.name
inner join ingredient_id on ti.ingredient = i.name

--insert reviews from temp table
insert into recipe_made (recipe_id, username, date, rating)
select r.id, t.user, t.datemade, t.rating
from _temp_kaggle_reviews t
inner join _temp_kaggle_recipes tr on t.recipetempid = tr.id
inner join recipe r on tr.name = t.name;
```

Analysis for most paired ingredients:

```
select
    i1.name,
    i2.name,
    count(ri1.recipe_id)
from ingredient i1
join ingredient i2 on i1.id > i2.id
inner join recipe_ingredients ri1 on i1.id = ri1.ingredient_id
inner join recipe_ingredients ri2 on i2.id = ri2.ingredient_id
where ri1.recipe_id = ri2.recipe_id
group by i1.name, i2.name
```

```
order by count(ri1.recipe_id) desc
```

Analysis for user retention:

```
select *
from
   (select
      count(*) as interaction,
      username as uname
   from recipe_made
   group by username) as f
inner join users on uname = users.username;
```