DATABASES 1 PROJECT

GROUP G

Grand Gourmet

WILLIAM EKDAHL
OLIVER PALONKORPI
MINHUI ZHONG
MARCUS OTTERSTRÖM
ALEXANDER ARVIDSSON
FILIP SIVERTSSON

TABLE OF CONTENTS

INTRODUCTION

Group G's task was to build a relational database prototype for the fictional business entity Grand Gourmet. Additionally, the group was tasked with creating an entity relationship diagram (ERD), mock data for the database, a database schema, and a series of database queries.

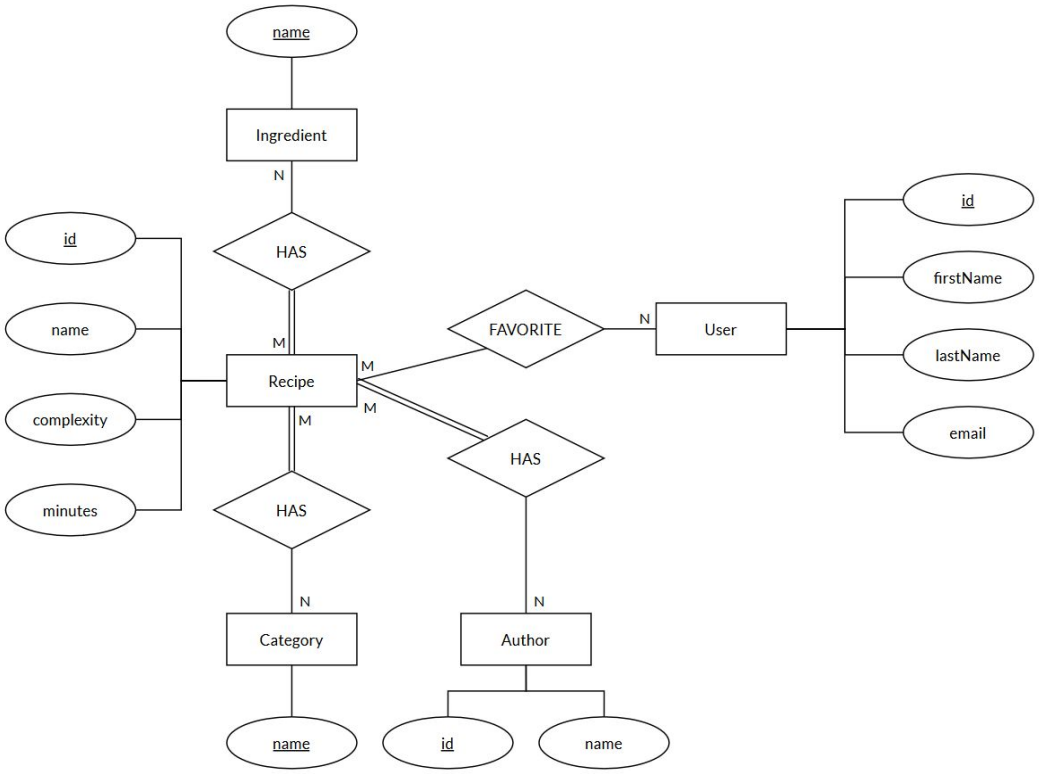
The prototype for Grand Gourmet's application had the following requirements:

- Each recipe should have one or more authors, categories, and ingredients.
- The steps required to cook each meal do not have to be stored in the initial prototype. Only the title, cooking time, and a complexity level (1-5) has to be stored in a recipe.
- Users should have a name and an email address, and should be able to mark recipes as favorites.

DATABASE

The database was implemented in SQLite version 3.33. To interact with the database, both the command line interface (CLI) for SQLite and DB Browser were used. Both the ERD and the database schema were made in draw.io.

ERD



Database schema

Recipe

id	name	complexity	minutes
1	Sesame pasta with bell	1	90
2	Olive salad with onion	3	45

RecipeCategory

recipeId	categoryName
1	2
2	1

RecipeAuthor

recipeId	authorId
1	1
2	1

RecipeIngredient

repiceId	ingredientName
1	pepper
2	pepper

FavoriteRecipe

userId	recipeId
1	2
2	2

Category

name
Christmas Food
Mexican Food

Author

id	name
1	Gordon Ramsay
2	Bobby Flay

Ingredient

name
salt
pepper

User

id	firstName	lastName	email
1	Jerry	Ternhag	ternhagjerry@yahoo.com
2	Carina	Lindahl	carinal3256@hotmail.com

QUERIES

Query 1

List all recipes that contain the ingredient 'tomato'.

Example result (61 rows in 7ms)

```
/* Author: Everyone */
SELECT Recipe.name
FROM RecipeIngredient
JOIN Recipe ON Recipe.id = RecipeIngredient.recipeId
WHERE RecipeIngredient.ingredientName = 'tomatoes'
GROUP BY Recipe.id
ORDER BY Recipe.name;
```

name
Almond puree with tomatoes
Cilantro salad with chicken
...

Query 2

List the names of all the recipes together with how many ingredients there are in each recipe.

Example result (1 000 rows in 4ms)

```
/* Author: Filip */
SELECT
    Recipe.name,
    COUNT(*) AS ingredients
FROM RecipeIngredient
JOIN Recipe ON RecipeIngredient.recipeId = Recipe.id
GROUP BY Recipe.id;
```

name	ingredients
Sesame pasta with bell	5
Olive salad with onion	8
...	

Query 3

List the names of all recipes and a column for complexity. For the complexity column, write "high" for those with difficulty level 4 or 5 and more than 5 ingredients, "low" for those with difficulty level 1 or 2 and less than 5 ingredients, and "medium" for the rest.

Example result (1 000 rows in 12ms)

```
/* Author: Filip */
SELECT
    Recipe.name,
    CASE
        WHEN Recipe.complexity > 3 AND COUNT(*)
> 5 THEN 'High'
        WHEN Recipe.complexity < 3 AND COUNT(*)
< 5 THEN 'Low'
        ELSE 'Medium'
    END AS complexity
FROM RecipeIngredient
JOIN Recipe ON RecipeIngredient.recipeId =
Recipe.id
GROUP BY Recipe.id;
```

name	compelxity
Parmesan dish with ginger	Medium
Lime dish with cilantro	High
Chili soup with turkey	Medium
Pineapple pie with stock	Low
...	

Query 4

List all available recipes that are tagged as 'Christmas food' and written by 'Jamie Oliver'.

Example result (30 rows in 13ms)

```
/* Author: William */
SELECT
    Recipe.name AS recipeName,
    Author.name AS authorName,
    RecipeCategory.categoryName AS category
FROM Recipe
JOIN RecipeAuthor ON Recipe.id =
RecipeAuthor.recipeId
JOIN Author ON Author.id =
RecipeAuthor.authorId
JOIN RecipeCategory ON Recipe.id =
RecipeCategory.recipeId
WHERE
    Author.name = 'Jamie Oliver'
AND
    RecipeCategory.categoryName = 'Christmas
Food'
ORDER BY Recipe.name;
```

recipeName	authorName	category
Almonds gravy with ginger	Jamie Oliver	Christmas Food
Banana casserole with lettuce	Jamie Oliver	Christmas Food
Beef pasta with cumin	Jamie Oliver	Christmas Food
…		

Query 5

List the ten recipes that have the most favorite selections.

Example result (10 rows in 1 328 170ms)

```
/* Author: Oliver */
SELECT
    Recipe.name AS recipeName,
    COUNT(*) AS amountOfFavorites
FROM FavoriteRecipe
JOIN Recipe ON Recipe.id =
FavoriteRecipe.recipeId
GROUP BY FavoriteRecipe.recipeId
ORDER BY COUNT(*) DESC
LIMIT 10;
```

recipeName	amountOfFavorites
Cheddar soup with broth	267 908
Almond pasta with potatoes	267 736
…	

Query 6

List the three users who have marked the most recipes.

Example result (3 rows in 730 321ms)

```
/* Author: Alexander */
SELECT
    User.firstName,
    User.lastName,
    COUNT(*) AS totalFavorites
FROM FavoriteRecipe
JOIN User ON User.id = FavoriteRecipe.userId
GROUP BY FavoriteRecipe.userId
ORDER BY COUNT(*) DESC
LIMIT 3;
```

firstName	lastName	totalFavorites
Tage	Rydberg	100
Thomas	Lundqvist	100
Bengt	Winkler	100

Query 7

List the category that has the most favorite recipes.

Example result (1 row in 1 164 781ms)

```sql
/* Author: Minhui */
SELECT
    RecipeCategory.categoryName,
    COUNT(*) AS favoriteCount
FROM RecipeCategory
JOIN FavoriteRecipe ON RecipeCategory.recipeId =
FavoriteRecipe.recipeId
GROUP BY RecipeCategory.categoryName
ORDER BY favoriteCount DESC
LIMIT 1;
```

categoryName	favoriteCount
Asian Food	39 414 432

Query 8

Calculate the average value for cooking time, degree of difficulty and number of ingredients for each author.

```sql
/* Author: Marcus */
SELECT
    Author.name,
    ROUND(AVG(Recipe.minutes), 2) AS averageMinutes,
    ROUND(AVG(Recipe.complexity), 2) AS averageComplexity,
    averageIngredientCount
FROM RecipeAuthor
JOIN Author ON RecipeAuthor.authorId = Author.id
JOIN Recipe ON RecipeAuthor.recipeId = Recipe.id
JOIN
(
    /* Average ingredient count per recipeAuthorId */
    SELECT
        recipeAuthorId,
        ROUND(AVG(ingredientCount), 2) AS averageIngredientCount
    FROM
    (
        /* Ingredient count per author and recipe */
        SELECT
            RecipeAuthor.authorId AS recipeAuthorId,
            RecipeIngredient.recipeId,
            COUNT(*) AS ingredientCount
        FROM RecipeIngredient
        JOIN RecipeAuthor ON RecipeIngredient.recipeId = RecipeAuthor.recipeId
        GROUP BY RecipeAuthor.authorId, RecipeIngredient.recipeId
    )
    GROUP BY recipeAuthorId
)
ON RecipeAuthor.authorId = recipeAuthorId
GROUP BY Author.id;
```

Example result (10 rows in 40ms)

name	averageMinutes	averageComplexity	averageIngredientCount
Gordon Ramsay	63.24	3.1	5.17
Jamie Oliver	61.37	3.16	5.43
…			

Query 9

List email to all users only favorite marked recipe that is without milk, cream, and cheese to be able to make targeted advertisements.

Example result (35 982 334 rows in 1 568 712ms)

```
/* Author: Everyone */
SELECT User.email
FROM FavoriteRecipe
JOIN User ON FavoriteRecipe.userId = User.id
WHERE User.id NOT IN
(
    /* Users with at least one favorite recipe containing
lactose */
    SELECT DISTINCT FavoriteRecipe.userId
    FROM FavoriteRecipe
    WHERE FavoriteRecipe.recipeId IN
    (
        /* Recipes with lactose */
        SELECT DISTINCT RecipeIngredient.recipeId
        FROM RecipeIngredient
        WHERE RecipeIngredient.ingredientName IN
        (
            'milk',
            'cream',
            'cheese',
            'butter',
            'parmesan',
            'yoghurt',
            'margarine',
            'cheddar'
        )
    )
)
GROUP BY User.id;
```

email
ines.olofsson@gmail.com
marbjo@live.com
alin.rosengren@gmail.com
jeanetteh@bredband.net
malin.sjoholm@outlook.com
martinliebe@hotmail.com
pettersson.paul1781@live.se
ebbe.helgesson@hotmail.com
petterssonfarbod@telia.se
texlindelof@bredband.net
olla.morley@gmail.com
…

Query 10

Recommend other recipes to a user based on what they have done for favorite selections and what other users have done for favorite selections.

```
/* Author: Everyone */
SELECT
    Recipe.id,
    Recipe.name,
    FavoriteRecipe.userId
FROM FavoriteRecipe
JOIN Recipe ON Recipe.id = FavoriteRecipe.recipeId
WHERE FavoriteRecipe.recipeId NOT IN
(
    /* Favorites from one user (user 5) */
    SELECT FavoriteRecipe.recipeId
    FROM FavoriteRecipe
    WHERE FavoriteRecipe.userId = 5
)
AND FavoriteRecipe.userId IN
(
    SELECT FavoriteRecipe.userId
    FROM FavoriteRecipe
    WHERE FavoriteRecipe.userId IN
    (
        /* Users with at least 2 same favorite recipes as user 5 */
        SELECT FavoriteRecipe.userId
        FROM FavoriteRecipe
        WHERE FavoriteRecipe.recipeId IN
        (
            /* Favorites from one user (user 5) */
            SELECT FavoriteRecipe.recipeId
            FROM FavoriteRecipe
            WHERE FavoriteRecipe.userId = 5
        )
        GROUP BY FavoriteRecipe.userId
        HAVING COUNT(*) > 1
    )
    AND FavoriteRecipe.recipeId NOT IN
    (
        /* Favorites from one user (user 5) */
        SELECT FavoriteRecipe.recipeId
        FROM FavoriteRecipe
        WHERE FavoriteRecipe.userId = 5
    )
    GROUP BY FavoriteRecipe.userId
    ORDER BY COUNT(*)
    LIMIT 2 /* Amount of recommendation users */
)
ORDER BY RANDOM();
```

Example result (3 rows in 124 155ms)

id	name	userId
393	Celery dish with onions	210 522
268	Basil gravy with pepper	1 433 150
257	Spinach salad with parmesan	173 281

WORK ANALYSIS

Group G (Minhui Zhong, Marcus Otterström, Alexander Arvidsson, Filip Sivertsson, Oliver Palonkorpi, William Ekdahl) focused on solving the assignment in a collaborative manner; most of the tasks were completed together. Despite this, a few tasks were split up and approached individually, and then discussed in group after they had been completed.

The first couple days consisted of the overall planning of the project. Decisions regarding how the group would communicate and what the schedule was going to look like were discussed and settled upon. The final schedule consisted mainly of two days, Mondays and Thursdays. If deemed necessary, the group would also meet on Fridays. Work hours were between 10:00-17:00 with the exception of an hour of lunch between 12:00-13:00. At the end of each session the group discussed what had been completed and what was left to see if additional time had to be allocated to stay aligned with the schedule. Most of the communication was through the instant messaging and voice call platform Discord.

During the first session the group planned the database structure and worked on the entity relationship diagram. The initial versions were finished within a couple hours. William then individually created the database schema in approximately 30 minutes, which was then analyzed by the group. These initial diagrams only required minor changes over the span of the project, which became apparent as the group worked with the database implementation in SQLite.

The database needed data, a task which was originally tackled by the entire group. Search engines were used to try and find relevant data on the internet in a usable format, for example various ingredients, recipes and chefs, preferably in a text file format. However, the group came to a quick realization that it would be very difficult to find the nische data that was required, and that it would be easier to generate the data and the SQL statements using some custom software that could be programmed internally. Since Marcus and Oliver were the most experienced in terms of programming, the task was split in half. Oliver built software to generate recipes, which included the title, complexity, cooking time, ingredients, categories, and author(s) of said recipes. Marcus worked on a user generator which would generate believable names and email addresses, as well as the recipe favorites of the users.

The recipe generator was created in approximately 4 hours by Oliver using Python. It can generate any amount of recipes using a number of ingredients, filling the recipe title with words such as "puree", "soup", "pasta", et cetera. It chooses the ingredients randomly, however, it includes sanity checks such as assuring that one recipe does not contain duplicate ingredients. The script also generates SQL statements for the category and the author(s) of the recipe.

The user generator was created by Marcus using C#. The creation of the generator took approximately 12 hours over the span of many iterations. The user generator uses real name statistics from Statistiska Centralbyrån (SCB) of all registered inhabitants of Sweden in 2019, including all first names and last names with at least 10 living people. This means that there are approximately 9 850 000 possible first names and 8 880 000 possible last names in the generator. However, it does not just randomly pick one of

the millions of first names and last names since that would not be a good representation of how names look in the real world. In the real world, "Andersson" is expected to be a much more common last name than "Bandelin", so the generator had to take the frequency of the names into account. Luckily, this data was also available at SCB and was used when creating the generation algorithm for the users. The user generator then uses the first name and the last name to create an arbitrary (but believable) email address for said user, making sure that the email had not been assigned to any previously generated user.

Group G was interested in generating as many users as possible, as there was an informal competition between other groups who would have the most amount of users in their database. Because of this, Marcus spent a lot of time trying to optimize the generator to work as fast as possible. Originally it was capable of generating and writing approximately 500 users to a text file per second. This meant that generating a large amount of users (in the millions) took a long time, so optimizations had to be found. Luckily, through optimizing the name frequency algorithm and batch writing to the text file, the generator was able to generate and write up to approximately 8000 users per second, which was deemed acceptable.

Using the generator, Marcus generated 100 000 000 users and ~266 000 000 favorites over the course of an afternoon (approx. 4 hours), just barely having enough available random access memory (RAM) to do so. However, when trying to execute the 15GB text file of SQL statements in SQLite, it was quickly found that inserting all 100 000 000 users would take over 1 000 hours at the rate it was going, leading to an investigation by Marcus on how to improve SQLite performance. It was quickly found that multiple INSERT- and UPDATE-statements should be batched together in transactions to drastically improve performance. The generator was modified to insert this into the output, along with some other minor performance improvements, which made the insertion of all 100 000 000 users possible in just under 90 minutes. However, as this database was highly inefficient to test queries against (some queries taking up to an hour to complete), another database was made alongside the original one with 100 000 users for testing and development purposes.

The next task was to write all the database queries. To do this, each person in the group was assigned one to two queries to write individually, leaving the last couple queries for the whole group. Not only was this done to be more time efficient, but also to make sure that everyone was highly involved in completing this task. This level of involvement would have been difficult to achieve if everyone had worked on each query together at the same time. In total, writing all the queries took approximately 17.5 hours. After everyone had finished their queries, the group gathered again and analyzed everyone's work together. During this session all queries were executed in the development database, both to ensure that they worked but also to make sure that everyone had a general understanding of how they worked. This was effective as everyone participated in giving constructive feedback on how to improve and/or fix the other group members' scripts, resulting in lots of improvements to all of the scripts.

The task of writing the documentation was tackled in the same manner as writing the queries. Each person was assigned one chapter to write, and after that was done, everyone in the group gave feedback on the text. Marcus then unified the document by writing the chapters with consistent tense and grammar. In total, the document took about 50 hours to write. Finally, Oliver single handedly worked on the presentation in PowerPoint, using information from the documentation. The rest of the group provided feedback, and translated the presentation to Swedish and started practicing presenting it together.

SECURITY, ETHICS, AND PRIVACY

Security, ethics, and privacy are all essential parts of designing and maintaining a database. The first security measure was to design the database with integrity constraints from the start to minimize the risks of invalid data being inserted into the database, which could potentially lead to loss and corruption of data. This included both key constraints (not null, unique, et cetera) and reference constraints (foreign keys). One exception to the enforcement of constraints was the unique many-to-many-but-at-least-one relationship between recipes, ingredients, authors, and categories, as seen in the ERD. Implementing these into the database would have been very impractical as it would have forced the recipe table to additionally contain one column for authorId, categoryName, and ingredientName, all with not null and foreign key constraints, to make sure that each recipe contains at least one of each. However, a bridge table would still be required for each type as most recipes have more than one ingredient, author, and category. Writing a query, for example to select all ingredients of a given recipe, would become much more complex. The relationships could possibly have been enforced using the CHECK-statement, but this was deemed to be too complex as well. Therefore, the group decided to not enforce this on a database level, but argued that it would not be a hard task for the backend developers to enforce on inserts and updates to the affected tables. However, because of the overall use of constraints, storing and changing data in the database became reliable and relatively safe, which increased the security of the database.

In terms of privacy, the group had to create a database that (at the very least) complied with the laws set forth in the General Data Protection Regulation (GDPR). According to the GDPR, personal data (such as names, address, etc) should not be collected more than necessary, and the data may only be used for specific and predetermined purposes. Additionally, sensitive data such as information that reveal ethnicity, political views, religious or philosophical beliefs, health, or sexuality shall not be collected under normal circumstances. It is also stated that the company should not keep any kind of personal data longer than necessary (Datainspektionen, 2019). The group is under the impression that the database generally fulfills the requirements of GDPR, however, with room for discussion. For example, collecting the full names of every user is not a vital part of the application. At the same time, if the group was to assign a random number for each user as their username, the application would feel a bit dull and impersonal. A good middle ground could have been to let each user assign themselves a username, like most modern applications do. In the real world, discussions would have been held with Grand Gourmet regarding the importance of storing the users' full names. Moreover, Grand Gourmet must make sure that all users agree to having their data collected and used in the ways that the company plans to, as well as clearly communicating how the users' data will be used. For example, query 9 assists in targeted marketing towards vegetarians and vegans by analyzing everyone's favorite recipes, which is a questionable use of the data in the database unless the users have consented to this. Grand Gourmet must also elect someone to be responsible for personal data and its use in the company (personuppgifts-ansvarig). In the real world, Grand Gourmet would also be responsible for securely storing the data and making sure that nobody unauthorized (both outside and within the company) is able to access the personal data of their users. As SQLite does not natively support users and permissions, which would be effective when protecting personal data from unauthorized Grand Gourmet employees, another database management system that supports this may be preferred.

REFLECTION

The group is happy with how the project went, both the result and the process. If there is one thing that could have been done differently, it would have been to take better notes of when time was allocated to the project as this would have made it easier to assess the time spent on each segment. If this task would have been approached in the real world, there would have been a lot of communication between the group and the employer (in this case, Grand Gourmet) to ensure that their requirements are fully met.

Everyone in the group wrote some queries and felt that they improved their ability to write and understand them. Since the database contained so many rows, the group got a better sense of the capabilities and limitations of SQLite, as well as why performance is important. One such learning was when Marcus tried to insert the person data, which would have taken over 1000 hours unless the performance could be improved. The group learned that the performance of inserting and updating many rows at once could be drastically improved by wrapping the statements in transactions. Some learnings by individual members of the group include how to better use constraints, that it helps to take breaks, how to plan and structure group work, and how larger databases are designed and created.

REFERENCES

Datainspektion, 2019. Enkla grunder i dataskydd.
Available at: https://www.datainspektionen.se/globalassets/dokument/enkla-grunder-i-dataskydd.pdf
[Accessed 16 December 2020].