Assignment 3 report

By Lisa Veltman, lv222fz

1. Idea

Problem:
I was pregnant in Spain last year and there was not a good app for pregnancy in Spain. Maybe one day I would like to develop it but to start, I am thinking about making a lighter version for this assignment and course.
I am thinking about making a website with some pregnancy information in English, for now, as the assignment has to be in English.

Users:
The main users will be pregnant woman and this website could be a perfect source for reliable information, written by midwifes and users.

Minimum features:
The website describes the pregnancy weeks.
Each week of pregnancy has a description written by one midwife or more. Each week has an image and related blogposts by other users. Each week is within a trimester that also has can have a description.
Each midwife and each user have a name, id, email etc.

Optional features:
This website can be developed and expanded.
If time permits, each blogpost can have categories or/and comments by other users. Local information about hospitals and clinics can also be added. The midwifes can then be connected to these.

Design:
I will need to have many tables and the design of the database is complex enough.
I also have to consider it to be easy to expand when designing.
I know how to build websites and would use php and mysql.

2. E/R Diagram

I decided to use “crows foot” notation in this diagram as I never used it before. I also found it better to display the difference in the relationships one-to-many versus zero-to-many.

First I started to sketch on paper and soon realized that each *week* could not have just an attribute with an Id of the author. I wanted the weeks to be able to have multiple authors, 1 or 20 for instance. It is not a good design to have redundant attributes. For example, it is not good to have attributes “author1”, “author2” and “author3”. A value of an attribute should only be one. It is bad design to have multiple values, for example “1, 2, 3”.
To solve this, I added the relationship “written by” with a notation *one or many-to-zero or many*.
The author had to be an entity set as it has more attributes, just as user.

The attribute *nr* of *week* could have been named *id* and then a *title* as the number of the week as a letter, but for now, a numerical title is good enough.

In my app each blogpost is connected to exactly one week. But if I wanted to expand the app and have blogposts that are related to none or many weeks, the arrow between *blogpost* and *week* would need to change.

As I stated in the section “idea” the app can be expanded and more entities sets would have to be added, but this diagram represents the minimum requirements. Some attributes are maybe not going to be used in the actual app, at the moment. Some attributes are there to be able to expand the app or because they make sense in another context. The same applies for missing attributes, when the app expands, more attributes would have to be added.

The attribute *email* in both *users*, and *authors* collections are intentionally not used as primary keys even if they should be unique. Instead, I added the attribute “id”. This is because there might be situations where you don’t want to display the email of the user. For example, let’s say I create links to view the user’s pages and have to be able to identify the user in the url. If I use the email as identifier, the email is exposed and the url becomes uglier.
Compare *users.php?id=1* to *users.php?id=lisa.student@linneus.com.*

3. Design in SQL

Design
The relationship *written by* becomes a relation because we have the relationship one-or-many-to-zero-or-many. As mentioned in previous section, we cannot have a column in the relation *weeks* called *authors* and have multiple values. Therefore, we need to add this relation. Compare this to the relationship *blogposts-users.* The blogposts are written by one user (and one user only), so we do not need to add a relation to store the user, it can be stored in a column in *blogposts*.

The attributes in the relation "written by" are the primary keys of the entity sets of the relationship, *id* and *nr*. To make this clearer I will rename them to *weeksNr* and *authorsId* in the new relation *writtenBy*.

I changed the names from the ER diagram to plural in the database, to make it clearer.

Except for the "written by" relationship, the rest of the relationships becomes attributes in one of the entities of the relation, connected by foreign keys.
The entity *blogposts* will have a column in the database table called *userId* and another one called *week*. These will be foreign keys linked to the tables *week* and *user* by their primary keys. This is possible as the relationships are many to one. The "one" allows us to connect as foreign keys. The same applies to the relationship *part of* between *week* and *trimester*. Trimester is the *one* side and therefore we can add its key as an attribute of *week*.

So, in the end, 6 tables will be added to the database:

- **authors(id, name, surname, email, about, image)**
  unique: email

- **blogposts(id, title, content, userId, created, updated, week)**
  foreign key: userId, week

- **trimesters(nr, title, description)**

- **users(id, name, surname, email, created)**
  unique: email

- **weeks(nr, trimestre, foto, content, created, updated)**
  foreign key: trimester

- **writtenBy(authorId, weekNr)**
  foreign key: authorId, weekNr

Normalization
To check if my design is good I applied normalization rules and checked if the design applied the most wanted form, BCNF. I say "most wanted" as it should be the form with less anomalies.

I checked the non-trivial functional dependencies of all relations. The relations *weeks, blogposts* and *trimesters* are all in BCNF as they only have the key on the left hand of the functional dependencies. In the case of *writtenBy* there are no non-trivial functional dependencies which makes is BCNF directly.

As I mentioned, I intentionally have email as a unique attribute in the relations users and authors. This way, they have functional dependencies that are not the key. In the case of users, id is the key and there is a functional dependency looking like "email->name, surname, id, created. The 2 relations are therefore not in BCNF but in 3NF as the pass the rest of the requirements for 3NF.

Each trimester should have a unique title and description but I will not set them as unique as it's a bad way to identify a row (identify a row by a text of 50 characters is bad) and there might be a point in the development where the data looks the same, "no description yet" or something similar.

4. SQL queries

I had to add more than the 5 required queries to be able to make the app work but I will only motivate the required once here. Some of the required queries might seem a little forced and I am totally aware of that there are many different queries that could give the wanted result back. Some multi queries can be exchanged to a query using JOIN, for example. I have used both methods to retrieve data from the database.

1. Multi query

To get the name and surname of the authors who wrote a specific week I used multi query. I retrieved data from two tables and only get data where the 2 tables matched in certain columns. Let's say we are looking at week 14 and I want to know who the authors are. I check in the table "writtenBy" and get all the rows that match weekNr 14. If that rows column *authorsId* matches the *id* in authors, I will get the name, surname and id from that author.

```
SELECT name, surname, id
FROM authors, writtenBy
WHERE authorsId = authors.id AND weeksNr = "14"
```

2. Multi query

I also used multi query to get all the info of a user and the titles of all the blogposts they have written. In the example below, I ask for all the titles of the blogposts that has the same value in the column *userId* and in the column id from the table *users*. I also added an “AND” to make sure I only got the titles from the wanted user.

```
SELECT * FROM users, blogposts

WHERE users.id = 2 AND users.id = blogposts.userId
```

3. JOIN

As I said, many times, INNER JOIN and multi query are similar, at least in my project. I also used INNER JOIN. This time I joined 2 tables to make sure only the posts and users that are in both tables are retrieved and not working with all the data from the 2 tables but the data I need. The join might be little forced in this case and can be substituted by other methods. I am using JOIN again, it can be seen under heading *5. View*.

```
SELECT week, title, name, surname, blogposts.id as blogpostId
FROM (blogposts
INNER JOIN users ON users.id = blogposts.userId)
ORDER BY week
```

4. Aggregation/grouping

At one point I am showing how many posts a specific user has written. To get a number I use the aggregate function COUNT(). This function counts how many times the given id is present in the table and this way we know how many posts that user has written.

```
SELECT COUNT(userId)
FROM blogposts
WHERE userId = 2
```

5. View

Instead of accessing all the data when working with the authors, we only need to work with some data and therefore create a view. In my case my app is still small and a view is not 100% necessary but as the app expands It can be useful for security reasons. I only work against the view in one query but in a bigger app, the view might be used more often.

I had to add “CREATE OR REPLACE” in my code so that the view would be created each time.
By joining tables I make sure only authors that has written on a week are displayed.

```
CREATE OR REPLACE VIEW authorsPublic
AS SELECT DISTINCT name, surname, id
FROM authors
INNER JOIN writtenBy ON writtenBy.authorsId = authors.id;
SELECT * FROM authorsPublic;
```

5. Implementation

I used php and mysql to develop the app. I am using a Mac and MAMP to run the website. I tested queries against the database in phpMyAdmin before inserting them in my source code. It was easier to view the results there. Once I had the desired result I used the query in my app and translated the result into html.

I implemented all required features and required queries for the assignment. I did not implement any optional requirements due to lack of time.
I focused on the queries and understanding the database for this assignment. My source code and design is weak and should be re-factored, I am totally aware of that.

Some content still needs to be created. For now, I am using the same image for all weeks and the descriptions are poor. Nevertheless, it does not change the implementation and use of the database. It is just a time-consuming task that I intentionally left out.

I would like to develop this idea fully but I need much more time and this assignment was a good introduction and I learned from it.

6. Video

The video can be viewed here: https://youtu.be/wW-7BRmFL8E