Database for Social Network

Documentation

Tamerlan Satualdypov and Aleksey Gusev

SE-2018. Astana IT University

Information and Communication technologies

Yerasyl Amanbek

Contents

Abstract

Our project work is represented by a database for a social network. This database contains all the necessary information for a social network with a wide range of functions.

As planned in the social network, you can create posts and entire groups, leave comments and put likes, attach bots to groups and send users private messages, etc. It is for these needs that our database is designed.

Our social network is more focused on building large communities than interaction between users. This is what we focused on when creating our database.

The main feature of our social network is bots that can be attached to many groups. This creates a great environment for creating diverse content and a good vibe in communities. Therefore, we consider our database to be a good example for creating a complete social network.

Introduction

Background

Before creating a database for a social network, we were faced with the task of identifying the characteristic features and characteristics that we will focus on. To do this, we have analyzed popular social networks, from which we have identified the most useful features that users like. In addition, it was necessary to read a variety of literature in order to understand the foundations of databases for social networks.

Importance of work

The importance of our work on creating a database for the social network was to create a database that would meet all the requirements and preferences of users. It also had to be convenient for maintenance and moderation by the developer. In addition, we wanted to create a database capable of further development and expansion. To do this, we had to create a solid foundation, eliminating all the smallest holes and shortcomings. We have put more emphasis on interaction in communities, because users are more active in interacting with a large number of people with common interests than in communicating with a narrow circle of people. Therefore, all our tables revolve around the main table with groups. This makes it possible to create useful tabs and create sorting by factors of interest.

Our second goal was to create an opportunity for daily user activity. On popular social networks, this activity is reflected in blogging, updating statuses and creating posts. In our social network, post is a versatile function for keeping active on a personal blog and communities. In addition, in order to meet popular criteria, we have made it possible to attach images in posts. Indeed, for current users, the ability to share their pictures and photos is irreplaceable.

Also, speaking of posts and blogs, we did not forget to add such a popular feature as rating with likes. It allows groups to see if others like what they are doing. For the same purpose, we made it possible to leave comments on posts. This allows a user to ask a question or create a discussion thread that any other user can connect to.

Our third goal was to create nice and useful features in the communities' section. For this, we added the ability to bind the bot to groups. The bot plays both an entertainment and a service role in our social network. It can be created and posted by communities to help create content and various features. It is also a kind of environment for the imagination of developers.

Our fourth goal was to create additional opportunities for sending private messages. It is present in any full-fledged social network, but in our work, we decided to limit ourselves to personal interactions between users. We also created the same function as in posts so that you can attach images to posts. This should have been enough for users to be able to find friends and start a personal conversation.

An important function for the administrator of the social network, which we decided to add first of all, was the ability to write reports on users who violated the rules or harmed other users.

Now, speaking about the possibility of displaying information and sorting it in detail, in the tables themselves, we have created separate records that can in this matter. In the user information, we added records with age, mail, and date of account creation. And in the group information, we added a verification record. The verification mark lets users know that a given group has been verified and that the content it creates is original. Also, with the development of the database, additional functions can be added to the verified groups.

Summing up the importance of our work, many may think that our project is more like a forum than a social network, but from our point of view, this is a plus, not a minus. After all, the more superfluous features and functions a user has, the less he focuses his attention on interacting and communicating with other users. And in popular social networks, this problem is urgent.

Methodology

To quickly design our DBMS and make less mistakes we looked at T. Connolly and C. Begg design guidelines[1]. So, our design methodology consisted of three phases:

- Conceptual – define entities, business rules and relations between them.
- Logical – provide details about the data and check whether the data model designed in conceptual phase structurally correct.
- Physical – table definition and implementation.

Each phase of this methodology helped us in designing well-structured and clean database.

Conceptual

At this phase we made steps related to conceptual database design. As said earlier, the main focus of this stage is to create a concept data model that fulfills all our requirements, which means that the core activity in this phase involves the use of ER modelling in which the entities, relationship and attributes are defined.

So, in this section we must have a data model that include:

- Entity types
- Business rules and Relationship types
- Attributes, primary and foreign keys

[1] From "Database Systems: A Practical Approach to Design, Implementation, and Management" by Thomas M. Connolly, Carolyn E. Begg, 2005.

Entity types

The first step to be performed is, of course, identifying the entity types, which are required for our data model. The information about entities is obtained from our specifications and requirements of a project.

In total, we have identified **ten** entities for the conceptual model:

`USERS, GROUPS, POSTS, POST_PHOTOS, LIKES, COMMENTS, MESSAGES, MESSAGE_ATTACHMENTS, REPORTS, BOTS`

Business rules and Relationship types

Business rules is a form of a constraint in the aspects of the database. They must be based on the way the database should act and requirements of a project.

We came up with the following statements:

- User could have zero or many groups subscribed
- Group must have at least one user as a subscriber
- Post could be with or without a photo
- User can set only one like to the one post
- User could write many comments to the one post
- Post could have zero or many likes
- User can receive many reports or never be reported
- Group can use zero or many bots
- Bots could be connected to zero or many groups
- User could send zero or many messages
- Message must have a sender and a receiver
- Message could contain an attachment

The relationships are typically described using a verb. They also must satisfy requirements of a project.

For our Social Network, we have identified the next relationships:

- USER **subscribes** GROUP (M:N)
- GROUP **creates** POST (1:M)
- POST **attaches** POST_PHOTOS (1:M)
- USER **sets** LIKE (1:1)
- POST **gains** LIKE (1:M)
- USER **writes** COMMENT (1:M)
- POST **receives** COMMENT (1:M)
- USER **receives** REPORT (1:M)
- GROUP **uses** BOT (M:N)
- BOT **connects** GROUP (M:N)
- USER **sends** MESSAGE (1:M)
- MESSAGE **contains** MESSAGE_ATTACHMENTS (1:M)

Attributes, primary and foreign keys

The last step of a conceptual design is to identify the attributes, primary and foreign keys for created entities.

For our project, we managed to give the following attributes for the entities:

USERS	**id** INT **nickname** VARCHAR(64) **email** VARCHAR(255) avatar_url VARCHAR(512) **registration_date** DATE

GROUPS	**id** INT **subscribers_id**^{FK} INT posts_id^{FK} INT bots_id^{FK} INT **name** VARCHAR(255) description TEXT
POSTS	**id** INT **owner_id**^{FK} INT photo_id^{FK} INT **title** VARCHAR(255) **text** TEXT **creation_date** DATE
POST_PHOTOS	**id** INT **photo_url** VARCHAR(512)
LIKES*	**user_id**^{FK} INT **post_id**^{FK} INT
COMMENTS	**id** INT **post_id**^{FK} INT **user_id**^{FK} INT **text** TEXT **creation_date** DATE
MESSAGES	**id** INT **sender_id**^{FK} INT **receiver_id**^{FK} INT attachment_id^{FK} INT **text** TEXT **creation_date** DATE
MESSAGE_ATTACHMENTS	**id** INT **photo_url** VARCHAR(512)

REPORTS	<u>**id**</u> INT **user_id**^{FK} INT **note** TEXT **creation_date** DATE
BOTS	<u>**id**</u> INT **nickname** VARCHAR(255) **description** TEXT

* – Table has a composite primary key; ^{FK} – Foreign Key; Underlined attributes stands for Primary Keys; Attributes in bold represents that they have NOT NULL constraint.

Logical

Our next step in designing database were performing all steps involved in logical phase. This stage's main focus is to translate data model from conceptual phase on to the logical structure and validate whether its able to support required transactions.

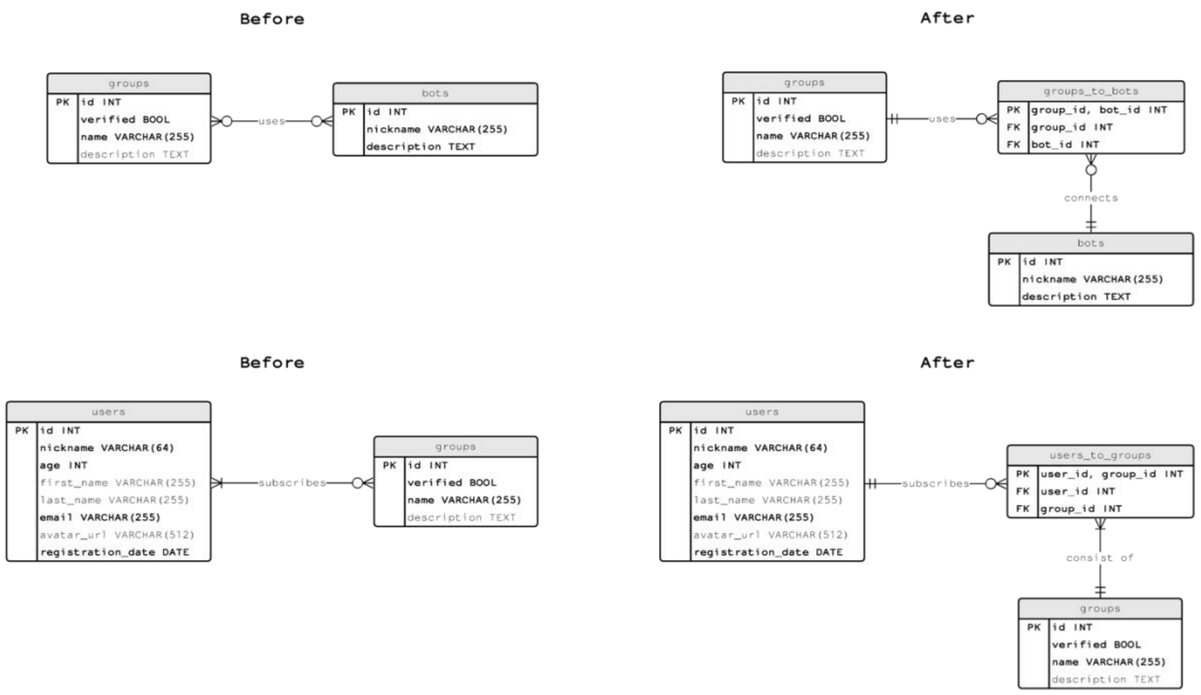
It turns out that we need to perform the following steps to transfer our database from a conceptual state to a logical one:

- Derive relations
- Validate using normalization
- Review data model

Derive relations

Looking at our conceptual model, we may notice several many-to-many relationships. Since our database is relational, we need to translate these relationships into one-to-many.

So to translate these relationships, we should create a table between them, this can be called a bridging table. After applying this technique to the `GROUPS to BOTS` and `USERS to GROUPS` relations we have following result:



After this transformation, our database looks more prepared to perform the required data transactions between tables.

Validate using normalization

Normalization is a popular technique when designing database. The main goal of applying this on to our database is to reduce the data redundancy and dependency.

There is a many forms of a normalization, but the most important is the first three normal forms:

- 1NF or First Normal Form – table must not have a repeating data, which means that when applying this normal form the data should be unique.
- 2NF or Second Normal Form – table must have one attribute as a primary key and the remaining columns should depend on it.
- 3NF or Third Normal Form – table must have columns that are non-transitively depend on the primary key.

The most important form from these three forms is the third one, because by the definition, in order for the model to be in the third normal form, tables should be in the previous normal forms too. So, at this step of translating data on to logical structure we mostly looked to validate tables is at the third normal form.

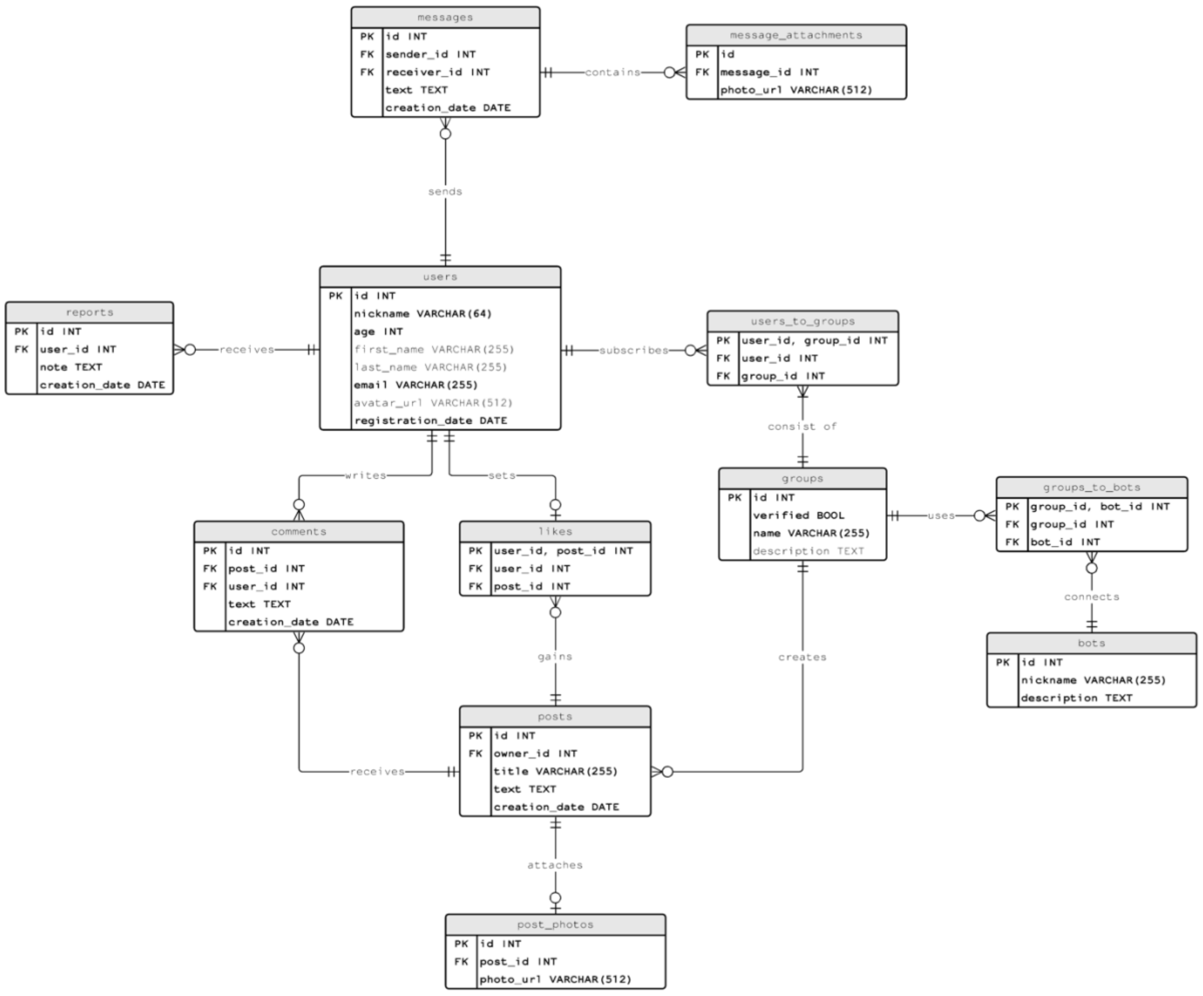
With the first and second form, everything is clear, but what does *non-transitevely* mean in the third form of normalization?

We made a little research on this question and found a good example[2]. In this example considered a `CUSTOMER` table with three columns: **`CustomerID`**, `CustomerZIP`, `CustomerCity`, where attributes depend on the primary key. With that design there could be a possible scenario when `CustomerZIP` attribute set to a zipcode of a different city without updating the `CustomerCity` which leads database to an inconsistent state. In order to fix this, they created a new table with two columns **`CustomerZIP`** and `CustomerCity` then added a foreign key

[2] Software Testing Help. (2020, November 13). Database Normalization Tutorial: 1NF 2NF 3NF BCNF Examples. Retrieved from https://www.softwaretestinghelp.com/database-normalization-tutorial/

in `CUSTOMER` table that references to a new table. That relation has a non-transitevely dependency and fixes an issues like they considered.

After understanding each normal form we applied them on to our database and finished with that result in the following ER-diagram:



Review data model

After completing the previous steps, we started reviewing our database, we had to make sure that the database allows us to perform all the intended transactions.

Fortunately, the resulting data model meets all our requirements: the user can write messages and comments, subscribe to groups and put likes on them, groups can publish posts, attach photos to them, and also connect bots.

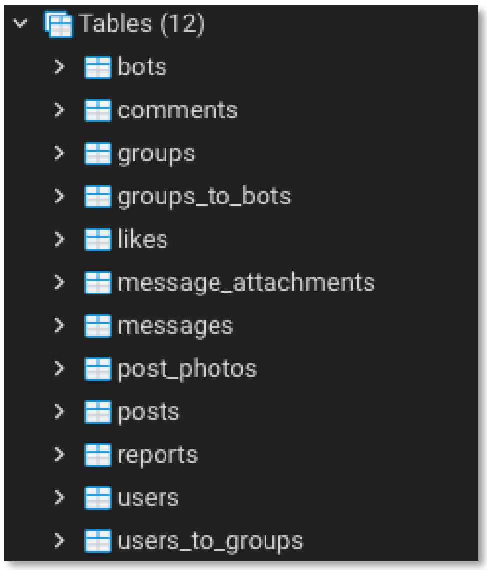
The logical phase helped us make the database ready for all tasks. These steps helped us establish the correct relationships between entities, as well as get rid of data redundancy and unnecessary dependencies.

Physical

At this stage we have a fully normalized entities after the logical phase. All we had to do is to translate these entities into actual database structure.

This means that we have to map:

- Entities to Tables
- Relationships to Foreign Keys
- Attributes to Columns
- Primary Keys to Primary Keys



```
Tables (12)
  bots
  comments
  groups
  groups_to_bots
  likes
  message_attachments
  messages
  post_photos
  posts
  reports
  users
  users_to_groups
```

So, after translating everything we have in total **twelve** tables connected depending on our ER-diagram initialization code could be found on GitHub.

Case study

Test case description

Let us consider the Social Network with the database that designed by applying our methodology. We could easily say that this Social Network will have a decent starter kit of abilities that user can perform on their website or mobile application. We can also add that the managers of this project may collect good analytics about the user's behavior and interests, as well as identify the target audience.

Users of the Social Network can be guests, as well as be registered. Guests can view posts and groups, but they can not like posts, subscribe to groups and write comments or messages. To register, users must specify their nickname and email address, which is used for validating the person. They could also provide their first name, last name and attach avatar to their profile, but this is not mandatory. Registered users are identified by unique identifiers. They have access to all the features that the Social Network provides.

Social Network has a messenger inside and all registered users could send their messages to each other. This gives an additional interactivity and attractiveness to the project, because users can make new friends and communicate immediately within the Social Network.

Another good option is that users of the Social Network can create groups that have access to the ability to publish different posts. Group administrators could also connect various of usefull bots to their groups that are available inside the Social Network. If it is a official page of a celebrity or an organization this group could get verified by project managers or moderators of the Social Network.

Test case queries

Here is the some interesting and simple queries that could be performed:

Query 1. Get name of every bot attached to a group:

```sql
SELECT groups.name, bots.nickname
FROM groups
INNER JOIN groups_to_bots ON groups.id = groups_to_bots.group_id
INNER JOIN bots ON groups_to_bots.bot_id = bots.id
ORDER BY name;
```

Query 2. Get only posts with attached photo:

```sql
SELECT posts.* FROM posts
INNER JOIN post_photos
ON posts.id = post_photos.post_id
ORDER BY posts.id;
```

Query 3. Get posts with more than 1 comment:

```sql
SELECT posts.* FROM posts
INNER JOIN comments ON posts.id = comments.post_id
GROUP BY posts.id
HAVING COUNT(*) > 1;
```

Query 4. Get posts that were made a year ago:

```sql
SELECT * FROM posts
WHERE EXTRACT(YEAR FROM posts.creation_date)=EXTRACT(YEAR FROM CURRENT_TIMESTAMP) - 1;
```

Query 5. Get statistics about amount of groups that uses each bot:

```sql
SELECT bots.nickname, COUNT(bots.nickname)
FROM bots
INNER JOIN groups_to_bots ON bots.id = groups_to_bots.bot_id
INNER JOIN groups ON groups_to_bots.group_id = groups.id
GROUP BY bots.nickname;
```

Query 6. Get all verified groups:

```sql
SELECT * FROM groups WHERE groups.verified;
```

Query 7. Get average age among registered users:

```sql
SELECT TO_CHAR(AVG(users.age), '99.9') AS average_age FROM users;
```

The following queries are more complex, but not less interesting:

Query 1. Get amount of messages sent and received for each user:

```
SELECT users.nickname, sent.count AS sent, received.count AS received
FROM users
INNER JOIN (
    SELECT users.id AS id, COUNT(*) AS count FROM users
    INNER JOIN messages ON users.id = messages.sender_id
    GROUP BY users.id
) sent
ON users.id = sent.id
INNER JOIN (
    SELECT users.id AS id, COUNT(*) AS count FROM users
    INNER JOIN messages ON users.id = messages.receiver_id
    GROUP BY users.id
) received
ON users.id = received.id;
```

Query 2. Get TOP-5 reported users:

```
SELECT users.* reports.count AS reports_count
FROM users
INNER JOIN (
    SELECT users.id AS id, COUNT(*) AS count FROM users
    INNER JOIN reports ON users.id = reports.user_id
    GROUP by users.id
) reports
ON users.id = reports.id
ORDER BY reports.count DESC
LIMIT 5;
```

Query 3. Get amount of subscribers that older than 18 for each group:

```
SELECT groups.*, adult_counter.count AS adult_count
FROM groups
INNER JOIN (
    SELECT users_to_groups.group_id AS group_id, COUNT(*) AS count
    FROM users_to_groups
    INNER JOIN (
        SELECT users.id AS id
        FROM users
        WHERE users.age > 18
    ) adult
    ON users_to_groups.user_id = adult.id
    GROUP BY users_to_groups.group_id
) adult_counter
ON groups.id = adult_counter.group_id
```

There are only 10 queries and this is only a small part of what can be performed using our database.

Results analysis

In the end, we have a very interesting, meaningful and important results. Our goal was to create a template design of a database for the Social Network so that it can be applied to any modern project.

We got a competitive Social Network with a good set of features: messaging, groups, bots, posting, likes and comments. People can easily use this Social Network and feel themselves comfortable.

The resulting database is easy to maintain, as we showed above, most queries are written easily, which is a good plus. The database is also extensible in any desired direction for a particular project.

We believe that the result of using the database we designed is successful and also believe that our work can be a starting point for a new projects.

Reflection

Satualdypov Tamerlan

What I did?

As I have some background in development of client-server applications I played a role of a kind of a team leader. Using my experience, I guided our team in a more correct practical direction.

My main contribution to our project was building the Entity-Relationship Diagram and its support, writing DDL and DML statements, twenty `SELECT` queries and subqueries, correcting the final results, designing the documentation and writing methodology and case study.

What I found challenging?

Since I mostly used NoSQL databases in my past developments, sometimes I made mistakes due to insufficient experience in designing relational databases.

Also, it was not very easy to start writing a methodology because of not knowing where to approach it correctly and where to start.

What I learned?

This project work gave me a very good experience in the development and design of relational databases close to real problems. Also, of course, this is a great experience in teamwork, which I am sure I will need in the future.

I consolidated the material I learned in ICT lessons, and became more confident in writing DDL and DML statements.

Gusev Aleksey

What I did?

I was responsible for tables with `GROUPS`, `POSTS`, `POST_PHOTOS`, `BOTS` and bridging table `GROUPS_TO_BOTS`. Also, I was developing DML statements and developed <u>ten</u> `UPDATE` and `DELETE` queries.

In the documentation work, I was engaged in the design of the abstract, introduction and conclusion and future work.

What I found challenging?

The difficult part for me was correctly write `DELETE` queries, because I had to clear the data that references to the table, which I want delete. Also I had some problems with joining table while I was trying to delete or update the data that was dependent on other records.

What I learned?

This project gave me a solid understanding of new techniques like: bridges, composite keys. I think now I will be able to use them in my future works. I also learned how to use `USING` operator instead of `JOIN` to delete data by criteria from other tables.

Conclusion

In conclusion, we would like to say that our project work can be turned into a good startup aimed at interaction between a large number of users. Our hybrid database takes the best from forums and social media, and strives to meet all the current needs of users and developers. While making the project, we tried not to copy and focus on some specific existing social network. Nowadays it is difficult to create something original, but there is an opportunity to incorporate the best of the existing into your work. We tried to fulfill this peculiar goal. In the end, we got what we wanted.

If we talk about how our project could be developed in the future, we already have a couple of new ideas. In the table we created, we could add features such as bookmarking selected posts, receiving notifications, and adding tags to groups. This would create more opportunities to search for communities by criteria of interest. Also, a nice feature to possibly add is to create custom icons that could be used in conversations and comments and are only available in the groups where they were created. This could be turned into a kind of commercial and creative content that would make communities even more special and interesting.

The flight of thought makes it possible to develop these tables and an unimaginable number, but so far, we have decided to limit ourselves to a solid foundation that can be developed in the future, and we believe that our work is available for further development.

References

Thanks for this resources for helping us to start and finish our project:

"Database Systems: A Practical Approach to Design, Implementation, and Management" by Thomas M. Connolly, Carolyn E. Begg, 2005.

DigitalDoughnut. (2018, June 4). *Social Media Databases: The Best Practices to Design a Working Database*. Retrieved from https://www.digitaldoughnut.com/articles/2018/may/social-media-databases-the-best-practices

Helen Vakhnenko, Agilie. (n.d.). *How to Create a Social Network Site from Scratch: Tips and Tricks*. Retrieved from https://agilie.com/en/blog/how-to-create-a-social-network-site-from-scratch-tips-and-tricks

Rampton, J. (2020, March 10). *How to Create a Powerful Social Network Platform in 8 Steps.* Retrieved from https://www.inc.com/john-rampton/how-to-create-powerful-social-network-platform-in-.html

Software Testing Help. (2020, November 13). *Database Normalization Tutorial: 1NF 2NF 3NF BCNF Examples*. Retrieved from https://www.softwaretestinghelp.com/database-normalization-tutorial/