# A PROJECT REPORT

on

# “Movie Database System using PHP and PostgreSQL”

Submitted to

**Technische Universität Chemnitz**

In fulfillment of the course subject

**Advanced Management of Data**

By

**Borneel Bikash Phukan (701125)**

**Daria Riakina (710077)**

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Faculty of Computer Science

Technische Universität Chemnitz

Chemnitz, Saxony - 09126

February 2022

# ABSTRACT

The goal of this project is to create a movie database system that is connected with a frontend user interface which enables a user to keep a track of all his movies, their ratings, any favorite stars associated with the movie. The user has the ability to manage the database by inserting new movie, star name, rating or even manage existing data by editing the data content, or even deleting the data content.

The frontend is made using basic HTML, and connected with the database schema by means of PHP. The database schema has been created using Postgresql and hosted on the university pgsql service. The initialization script has been attached with the project source code and a further file of the queries used for creating has been attached in order to prevent further changes to the database.

The project contains multiple dummy data which includes dummy movies, dummy people related with the movie, and even dummy ratings which can be fetched, edited or even deleted as per the will of the user. Our focus is on the functioning of the important components of the project so thereby no login credentials have to be provided to access the database from the php page. The project can be run straight from the script.

**Keywords**: PLSQL, PHP, Recommender System, Movie Database

# CONTENT

# INTRODUCTION

The project is intended for users willing to keep track of a personalized database of the movies watched and willing to watch. The entire project is divided into four sections namely suggested movie section, movie section, related person section and rating section.

The suggested movie section consists of a table that shows all the suggested movies which are based on the movies previously rated by the user. The movies are immediately fetched from the database as soon as the user accesses the project. This is done to showcase the dynamicity of the project.

The movie section consists of a hidden table and a panel of buttons with different functionalities. The work of the table is to show all the existing movies present in the database. The attributes of each movie are title, release year, minimum age of viewers, genre(s), rating, country of origin and language of the movie. The panel below the table includes a form which enables the users to provide inputs for inserting movies. Other buttons in the panel include SHOW ALL which enables to show all the movies present in the database, CHANGE FILM button renders a different page which consists of a form where the movie data can be updated. The REMOVE FILM removes a film that the user mentions in the textbox near it.

The person section consists of a hidden table and a panel of buttons. The work of the table is to show all the existing movie related people in the database. The person table consists of the person's name, the date of birth of the person, gender, role and the title of the movie performed. The form below it consists of all the input fields for inserting data into the table and the button panels includes buttons SHOW ALL, CHANGE PERSON and REMOVE PERSON which fetches all the related persons, opens another tab for updating data and remove mentioned person from the database respectively.

The final Rating section consists of a form to input movie name requiring rating and a rating field for rating input and an ADD RATING button. Similarly the CHANGE RATING button and an input textbox allows updating of rating, SHOW ALL shows all movies and its rating in the database and REMOVE RATING removes rating of the movie mentioned in the adjacent textbox.

# WHY USE PHP FOR THIS PROJECT

Being the most versatile server-side scripting language, the primary reason behind using it for our project is the ease of deploying it into our frontend and its simple features configured for PGSQL which helps to take user input and directly make the changes in the database. The programming language being one of the first server-side languages, has a large community which makes learning and debugging problems much easier. As size constriction was also a major factor for the project, we decided to refrain from using package managers like npm or pip. Also since the main intention of the project is the database, we decided not to use any fancy modules and straight away focus on the goals of the project.

Regarding the programming language to develop a dynamic website, a comparative analysis was conducted, which revealed the most appropriate programming language (Table 1).

| Name | Essence | Pros | Cons |
|---|---|---|---|
| PHP | A server-side language, for creating Web-sites Web-development orientation | ● cross-platform<br>● free<br>● easy to learn. | ● narrow profile.<br>● security.<br>● inconsistencies in the code. |
| JAVA | Cross-platform portable software code for the Internet | ● flexibility<br>● object-oriented programming<br>● java syntax is based on c ++<br>● platform independence<br>● multithreading<br>● security | ● Paid commercial use<br>● Low performance<br>● Lack of native design<br>● Multiple words and complex code. |
| Python | Universal high-level language, focused on improving developer productivity and code readability open development | ● fairly easy to learn, especially at the initial stage;<br>● syntax features encourage the programmer to write well | ● modularity mechanisms are well-designed and can be easily used. compilation slowness<br>● not an ideal language for programming multi- |

| | | | |
|---|---|---|---|
| | | readable code<br>● provides tools for rapid prototyping and dynamic semantics<br>● has a large community, positively disposed towards newcomers?<br>● many useful libraries and language extensions<br>● modularity mechanisms are well-designed and can be easily used. | threaded applications with a high degree of parallelism.<br>● not many qualities software projects;<br>● inherently limited means for working with databases. |

*Table 1 - Comparative Characteristics of Programming Languages*

Let us summarize the analysis. Although to date Java, C++ are the undoubted leaders in popularity. But they are not devoid of disadvantages. Therefore, as a programming language for modeling automated systems we chose PHP.
PHP - the best language for creating dynamic Web pages, for this language developed a huge number of frameworks.

In order to run the project and successfully connect to the database, it is necessary that the device has php installed and the extensions of pgsql are enabled. To do that simply uncomment the pgsql and pdo extensions in the php.ini file in the php directory. This enables the pg_connect() and pg_query() to connect with the database and process all the PGSQL queries. The project has been tried and tested many times and any errors associated with the database connection have been thoroughly eliminated. Thanks to the extensive functionalities of PHP, the database shall be immediately connected and running in the background as soon as the project has been started in the localhost server.

# PROJECT PLANNING

The planning of this project began with the design intuition of the database schema. Once with the completion of the database schema, we proceeded to design the frontend with an explicit focus on handling all the inputs and fulfilling all the criteria given in the task. Every input is handled by means of forms and each form is directly linked to the database using PHP. The entire frontend is divided into three sections as mentioned before in order to express the segregation of tasks mentioned in the question.

Since we are using different pages for altering data in the first two sections, it was deemed mandatory to separate the code for connecting to the database as we were calling the connect database code segment repeatedly. The file dbConnect.php fulfills the purpose of connecting to the database. It consists of the connection credentials like database name, port, password etc. and makes sure that every time we load a page, it seamlessly connects to the database.

The schema design was the most challenging part of the project planning. We interpreted and implemented the database schema in the phpMyAdmin service provided by the university and connected with the project using PHP. In the end, we came up with the following project structure:

| Project | | |
|---|---|---|
| 1. | CSS | |
| | 1. | style.css |
| 2. | alterMovieData.php | |
| 3. | alterPersonData.php | |
| 4. | dbConnect.php | |
| 5. | index.php | |
| 6. | database.sql | |
| 7. | readme.txt | |

# DATABASE SCHEMA

As for the database, before we started creating it, we developed a UML diagram showing all the entities, attributes of entities and relationships between entities. The diagram is shown in the figure below:

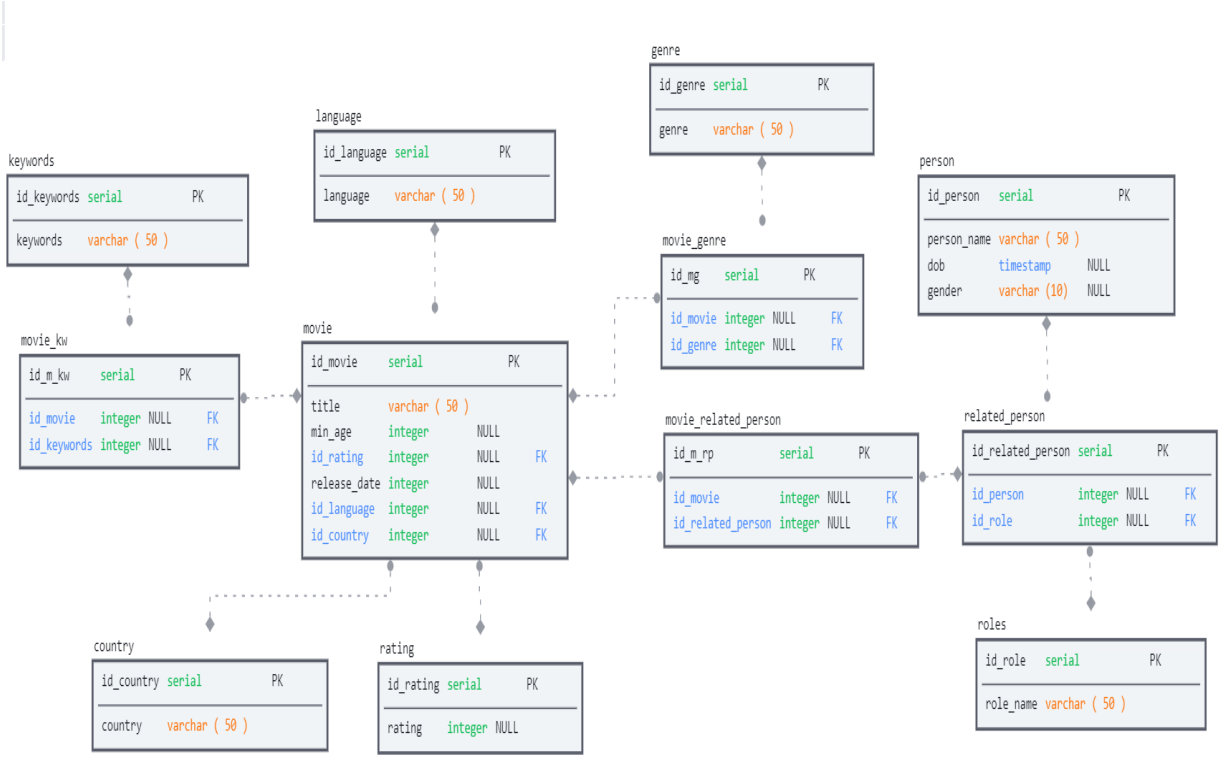


*Figure 1: UML Diagram of the Database*

Based on the diagram presented, a database was developed in Postgresql using the web interface phpPgAdmin. The database script is given in Appendix 1.

- **Database connection and manipulation**

  The database is connected to the frontend using the dbConnect.php script. The database is connected and is up and running in the background as soon as the user interface is loaded. The buttons in the user interface has flawless access to the database and can manipulate data as per the user's wish. Since we are using basic PHP without AJAX, every button click will reload the whole page instead of sections.

- **Table Relationships**

  The various tables that are being used in the schema has relationships amongst them like m:1 or m:n. This section describes the relationship in brief.

| Entities | Definition | Relationship |
|---|---|---|
| Movie (n) AND country (1) | One movie can be released by more than one country, one country only releases many movies. | 1: n |
| Movie(n) AND language (1) | Many films are produced in one language. One movie is released in one language. | 1: n |
| Movie (m) AND genre (n) | One movie may have more than one genre, several movies are released in the same genre | m: n |
| Person (m) AND roles (n) | One person can have multiple roles. One role can have more than one person | m: n |
| Movie (m) AND related_person (n) | One movie can have more than one person, one person can be in more than one movie | m: n |
| Movie (n) AND rating (1) | One movie has one rating, the same rating can have multiple movies | 1: n |
| Movie (m) AND keywords (n) | One movie can have several keywords. One word can contain more than one movie | m: n |

- **Description of the database schema**

  The entire database consists of multiple tables each having a particular purpose in the project. The primary tables that we have created are movie, person, roles, genre, rating, country, language, genre and keywords.

  Since tables like movie and genre, movie and keywords, movie and persons have many to many relations, we created some intermediary tables like movie_genre, movie_kw, movie_related_person and related_person.

The movie table has the attributes title, min_age, release_date, id_language, id_country and id_rating. The last three attributes are foreign keys to the tables language, country and rating respectively.

For person, the table related_person has two foreign key attributes id_person and id_role which references the person table and roles table. The person table has the attributes person_name, dob, and gender. The roles table has the attribute role_name and stores the values Director, Producer and Actor. The table related_person is again referenced by the table movie_related_person whose task is to relate movie related person to the movie.

For genre of the movies, we decided to create a genre table with pre-defined genres (Comedy, Action, Fantasy, Drama). The user can select multiple genres for a single movie as well. The genre table is referenced to the movie table with the help of movie_genre intermediary table.

For the recommender system, we needed keywords which assists the recommender system in making accurate suggestions. Therefore, we created a table called keywords with pre-existing keywords namely Hero, School, Blockbuster, Epic, Vampire, True Love, Teen Movie and Crime. These keywords are used by the recommender system for proper recommendation. The keyword for each movie is referenced by the intermediary table movie_kw.

# RECOMMENDER SYSTEM

## Algorithm

The recommendation system is based on sql-queries, where unrated movies with similar attributes to those of movies previously watched by the user are offered as recommendations.

To develop the Recommender system, it is necessary to:

1. Develop a database.
2. Create an interface that allows the user to work with the system.
3. Organize the interconnection and the ability to manipulate information.

The steps that we implemented in developing the recommender query are given below:

1. The parameter attributes for recommendations are genre, keywords, and actors/ directors.
2. The query first selects all the movies that the user has not rated, including the previously selected parameters.
3. The next step selects the parameters of already rated movies with a score greater than or equal to 3.
4. Finally, the parameters of unrated movies are compared to those of rated movies.
5. As a result, we get the selected movies, whose parameters correspond to the movies already watched by the user.

The query for the recommender system is given below:

```
SELECT movie.title FROM movie
LEFT JOIN movie_genre ON movie.id_movie=movie_genre.id_movie
LEFT JOIN genre ON movie_genre.id_genre=genre.id_genre
LEFT JOIN movie_kw ON movie.id_movie=movie_kw.id_movie
LEFT JOIN keywords ON movie_kw.id_keywords=keywords.id_keywords
LEFT JOIN movie_related_person ON movie.id_movie=movie_related_person.id_movie
LEFT JOIN related_person ON movie_related_person.id_related_person=related_person.id_related_person
LEFT JOIN person ON related_person.id_person=person.id_person
LEFT JOIN roles ON related_person.id_role=roles.id_role
```

```
LEFT JOIN rating ON movie.id_rating=rating.id_rating WHERE (
movie.id_rating IS NULL AND movie_genre.id_genre IS NOT NULL
AND (movie_genre.id_genre IN (
SELECT genre.id_genre FROM movie
LEFT JOIN movie_genre ON movie.id_movie=movie_genre.id_movie
LEFT JOIN genre ON movie_genre.id_genre=genre.id_genre
LEFT JOIN rating ON movie.id_rating=rating.id_rating WHERE (
movie.id_rating>3 AND movie_genre.id_genre IS NOT NULL)) OR
movie_kw.id_keywords IN (
SELECT keywords.id_keywords FROM movie
LEFT JOIN movie_kw ON movie.id_movie=movie_kw.id_movie
LEFT JOIN keywords ON movie_kw.id_keywords=keywords.id_keywords
LEFT JOIN rating ON movie.id_rating=rating.id_rating WHERE (
movie.id_rating>3 AND movie_kw.id_keywords IS NOT NULL)) OR
movie_kw.id_keywords IN (
SELECT keywords.id_keywords FROM movie
LEFT JOIN movie_kw ON movie.id_movie=movie_kw.id_movie
LEFT JOIN keywords ON movie_kw.id_keywords=keywords.id_keywords
LEFT JOIN rating ON movie.id_rating=rating.id_rating WHERE (
movie.id_rating>3 AND movie_kw.id_keywords IS NOT NULL)) OR
movie_related_person.id_related_person IN (
SELECT related_person.id_person FROM movie
LEFT JOIN movie_related_person ON movie.id_movie=movie_related_person.id_movie
LEFT JOIN related_person ON movie_related_person.id_related_person=related_person.id_related_person
LEFT JOIN person ON related_person.id_person=person.id_person
LEFT JOIN roles ON related_person.id_role=roles.id_role
LEFT JOIN rating ON movie.id_rating=rating.id_rating WHERE (
movie.id_rating>3 AND movie_related_person.id_related_person IS NOT NULL))))
GROUP BY movie.title LIMIT 5;
```

The query ensures that a maximum of 5 suggested movies are fetched at a time from the database.

# USER GUIDE

The following content guides the user how to use the product and what to expect from each button given in the user interface. To open the project, it is important that the php server is up and running in the background. To do that, ensure that php is installed in the computer along with postgresql. The command to run the php server is given below:

*php -S localhost:3000*

Open localhost:3000 and you will find the entire project up and running.

The database is already set up on the university phpmyadmin and all the credentials of the database are provided in the dbConnect.php. In case of any verification, the script for the database is attached in the zip file.

- **Your Movies Section:**

**ADD MOVIE** – This section is present inside a form where the user can input all the entities that are required to define a movie. All the data related to a movie is inserted in this section and this button is used to add the movie into the database.

**SHOW MAIN MOVIES** – This button is used for showing individual movies in the database. In case a movie has sub-parts, these will not be shown and will remain hidden from this section.

**SHOW SUB MOVIES** – This button is used for showing the sub-parts of a particular movie. The sub-movies that were hidden on clicking the first button will be shown here.

**CHANGE FILM** – This button redirects the user to a different page where the user will be presented with a form where the name of the film that needs to be changed should be put and all the updated movie data should be entered. Upon clicking the submit button, the film data inside the database gets changed.

**REMOVE FILM** – The remove film button is used to remove a movie from the database. In case the movie has more than one sub-part or sub-movies, then those movies will be removed from the database as well.

- **Person Section:**

**ADD PERSON –** This button is linked with a form that takes all the details regarding a person related to the movie.

**CHANGE PERSON –** This button redirects the user to a different page where we have a form where the person whose details need to be changed

**REMOVE PERSON –** This button removes a movie related person from the database. A person can be related with multiple movies at a time and when a person is removed, all the movies associated with the person is also removed as well.

**SHOW ALL –** This button is used for showing all the related persons (actors, directors and producers) and the movies associated with the person. In case there is more than one movie the person has worked on; the name of the person will be visible twice on the table with different movie name in the movie column.

- **Rate your Movies:**

  **ADD RATING** – This button is used for rating any of the unrated movie inside the database. The button is connected to a textbox which will only take the name of unrated movies and the rating by the user, and on clicking this button, it will add the rating to the unrated movie.

  **CHANGE RATING** – This button is used to change or alter the existing rating of a movie. The button is connected to a textbox which takes the name of the movie whose rating needs to be changed and a user rating.

  **SHOW ALL** – This button is used for showing all the ratings of the movies in the database.

  **REMOVE RATING** – This removes the rating of a movie from the database.

# CHALLENGES & SOLUTIONS

Although the project was successfully completed according to the requirements put forth, it did not come without challenges. There was countless minor to major challenges that were faced during development, and each were countered with critical thinking. This section focuses on addressing some of the major challenges that we faced while manipulating the database schema and how we overcame them.

**Retrieving data from multiple tables:**

The schema contains multiple tables and by looking at the schema diagram on Figure 1, it can be concluded that retrieval of information from different tables is necessary. To do that, we used LEFT JOIN on the tables whose data we need to retrieve and showcase on the frontend. For example, while retrieving the movies (not sub-movies) from the database, we need to retrieve all the data from movie table, rating table, genre table, language, and country table. Therefore, in this case, we first select the entities required and perform LEFT JOIN on each of the tables that it belongs to. Given below is the query for better explanation:

```
SELECT movie.title, movie.release_date, movie.min_age, rating.rating,  string_agg(genre.genre, ', ') AS
Genre,language.language, country.country
FROM movie LEFT JOIN movie_genre ON movie.id_movie=movie_genre.id_movie
LEFT JOIN genre ON movie_genre.id_genre=genre.id_genre
LEFT JOIN rating ON movie.id_rating=rating.id_rating
LEFT JOIN language ON movie.id_language = language.id_language
LEFT JOIN country ON movie.id_country = country.id_country
WHERE movie.id_movie NOT IN (SELECT id_movie_2 FROM related_movies WHERE movie.id_movie=related_movies.id_movie_2)
GROUP BY title, language.language,country.country, movie.release_date, movie.min_age, rating.rating;
```

**Show movies and Sub-movies:**

Sometimes a movie can have multiple parts within it. For example Harry Potter and Lord of The Rings are some of the movies which have multiple parts within it like Harry Potter and the Philosopher's Stone, Harry Potter and the Chamber of Secrets etc. These we are referring to as sub-movies. Since the requirement was such that the sub-movies should not be shown at first and only be shown on the second step, we decided to create a second button called SHOW SUB-MOVIES which will simultaneously show all the sub-movies under the main movie.

The queries for showing movies and sub-movies are given below:

| SHOW MOVIES | SHOW SUB MOVIES |
|---|---|
| SELECT movie.title, movie.release_date, movie.min_age, rating.rating, string_agg(genre.genre, ', ') AS Genre,language.language, country.country<br>FROM movie LEFT JOIN movie_genre ON movie.id_movie=movie_genre.id_movie<br>LEFT JOIN genre ON movie_genre.id_genre=genre.id_genre<br>LEFT JOIN rating ON movie.id_rating=rating.id_rating<br>LEFT JOIN language ON movie.id_language = language.id_language<br>LEFT JOIN country ON movie.id_country = country.id_country<br>WHERE movie.id_movie NOT IN<br>(SELECT id_movie_2 FROM related_movies WHERE movie.id_movie = related_movies.id_movie_2)<br>GROUP BY title, language.language,country.country, movie.release_date, movie.min_age, rating.rating | SELECT movie.title, movie.release_date, movie.min_age, rating.rating, string_agg(genre.genre, ', ') AS Genre,language.language, country.country<br>FROM movie<br>LEFT JOIN movie_genre ON movie.id_movie=movie_genre.id_movie<br>LEFT JOIN genre ON movie_genre.id_genre=genre.id_genre<br>LEFT JOIN rating ON movie.id_rating=rating.id_rating<br>LEFT JOIN language ON movie.id_language = language.id_language<br>LEFT JOIN country ON movie.id_country = country.id_country<br>WHERE movie.id_movie IN(SELECT id_movie_1 FROM related_movies WHERE movie.id_movie=related_movies.id_movie_1) OR movie.id_movie IN(SELECT id_movie_2 FROM related_movies WHERE movie.id_movie=related_movies.id_movie_2)<br>GROUP BY title, language.language,country.country, movie.release_date, movie.min_age, rating.rating ORDER BY soundex(title), release_date"; |

**Inserting data into multiple tables simultaneously:**

In situations such as inserting movie or inserting person data it was necessary to insert into different tables while also maintaining the flow of query executions. This was made possible using PHP function pg_query() which enabled to execute postgresql queries on the go.

**UI input related issues:**

- **Inputs from checkboxes:**
  Taking inputs from checkboxes was a very tiring procedure as iterating through every checked checkbox required us to run the query repetitively. This we could achieve by considering the checkbox elements as an array of elements and using implode() function to convert it to string.
- **Dynamic table to show data:**
  It is necessary that to make the user interface clutter free, we needed to make use of a table that shows data and in other times it stays hidden. Therefore, this type of table is only visible on button click when a certain action is being carried out. This we could achieve by calling the table with data inside when we invoke any of the buttons.

# CONCLUSION

The focus of the project is to develop a database and apply all the functionalities of Postgres to carry out all the necessary retrieval, altering and deleting of data from the database. The schema of the database has been designed in such a way that the schema can be extended to add more functionalities to the movie database

There were also focuses given on the designing and implementation of the recommender system. We have based the criteria for recommendation on genre, roles and rating. Every time a user sets a rating, it compares the parameters of the rated movie and suggests the unrated movies in the suggestion table.

In case of arranging the sub movies according to the names, we have used a postgresql library called ‘soundex’ which takes note of the sound of the word when spoken in English. It then arranges the sub-movies in alphabetic sequence, and we have a well-defined list of sub-movies.

Therefore, overall, the project was a highly successful one and we have covered all the basics that were required to implement the project. We have attached all the necessary documentations, files, and some dummy data inside the database for demonstration purposes and we really hope that the project consists of all the requirements that were asked.

# REFERENCES & BIBLIOGRAPHY


1. McKinney W. Python and Data Analysis [Text] / W. McKinney, translated from English by A.A. Slinkin. - Moscow: DMK Press, 2020. - 540 p.: ill.

2. Lopatin V. M. Informatics for engineers: Tutorial. [Electronic resource] / V. M. Lopatin SPb: Lan' Publisher, 2019. 172 p.: il Access: https://e.lanbook.com/reader/book/115517/#1 (date of reference: 29.01.2022).

3. https://www.postgresql.org/docs/current/ - Postgresql documentation.

4. https://stackoverflow.com/questions/4331353/retrieve-data-from-db-and-display-it-in-table-in-php-see-this-code-whats-wron#:~:text=php%20%24connect%3Dmysql_connect(',(%24result)%7B%20echo%20%22%3C – Dynamic table in user interface.

5. https://habr.com/ru/post/488054/ - Understanding relationship between tables.

6. https://docs.microsoft.com/en-us/sql/t-sql/functions/soundex-transact-sql?view=sql-server-ver15 – Soundex documentation.

7. https://www.w3schools.com/php/ - PHP Tutorial

8. https://www.php.net/manual/en/book.pgsql.php - PHP and PGSQL manual for development.

# INDIVIDUAL CONTRIBUTIONS TO THE PROJECT

## Borneel Bikash Phukan (701125):

Contributed to the development of the frontend, the project structure, linked the database with the frontend user interface and testing, logic implementation and execution of the postgresql queries in php environment. Ensured every php page relates to one another and flawless data fetching, manipulation and transmission is not hindered due to logical or syntactical error. Also contributed to the documentation and presentation of the project.

## Daria Riakina (710077):

Contributed to the development of the database schema with the development of the query for the recommender system. Wrote, tested and ensured that every query performs the desired task and the data stored in the database is well structured and well presented. Also contributed to the documentation and presentation of the project.