|  |
| --- |
| Logical DB Design  **Poll sharing system** |
| **Obraz zawierający róg, kreskówka, rysowanie, ilustracja  Opis wygenerowany automatycznie** |

Contents

[1 Business Description 3](#_Toc62212630)

[1.1 Business background 3](#_Toc62212631)

[1.2 Problems. Current Situation 3](#_Toc62212632)

[1.3 The benefits of implementing a database. Project Vision 3](#_Toc62212633)

[2 Model description 3](#_Toc62212634)

[2.1 Definitions & Acronyms 3](#_Toc62212635)

[2.2 Logical Scheme 3](#_Toc62212636)

[2.3 Objects 3](#_Toc62212637)

# 

# Description

## background

A user wants to create a poll that can be freely shared with a group of friends, who should also be able to forward it to others. The poll must support multiple-choice answers, and the creator should have access to real-time results.

## Problems

Given the four-week timeline to deliver core functionality, the database must stay lightweight and straightforward, yet flexible enough to support future scalability.

## DB Project Vision

Implementing a structured database provides a solid foundation for managing poll-related data efficiently and securely. It ensures data consistency, enables fast querying, and supports the integrity of user inputs and poll results. By centralizing information such as user-generated polls, answer options, votes, and access tokens, the system can offer a seamless and reliable experience for both poll creators and participants.

From the project vision perspective, the database is designed to support a minimal viable product (MVP) within four weeks while maintaining scalability for future expansion. This includes the potential to add features such as authentication, poll analytics, and user management without major architectural changes. A well-designed database schema allows for modular development and ensures that new features can be integrated with minimal friction.

In summary, the database is not only a storage component but a strategic backbone of the entire system, enabling real-time interaction, secure data sharing, and future-proof growth.

# Model description

## Definitions & Acronyms

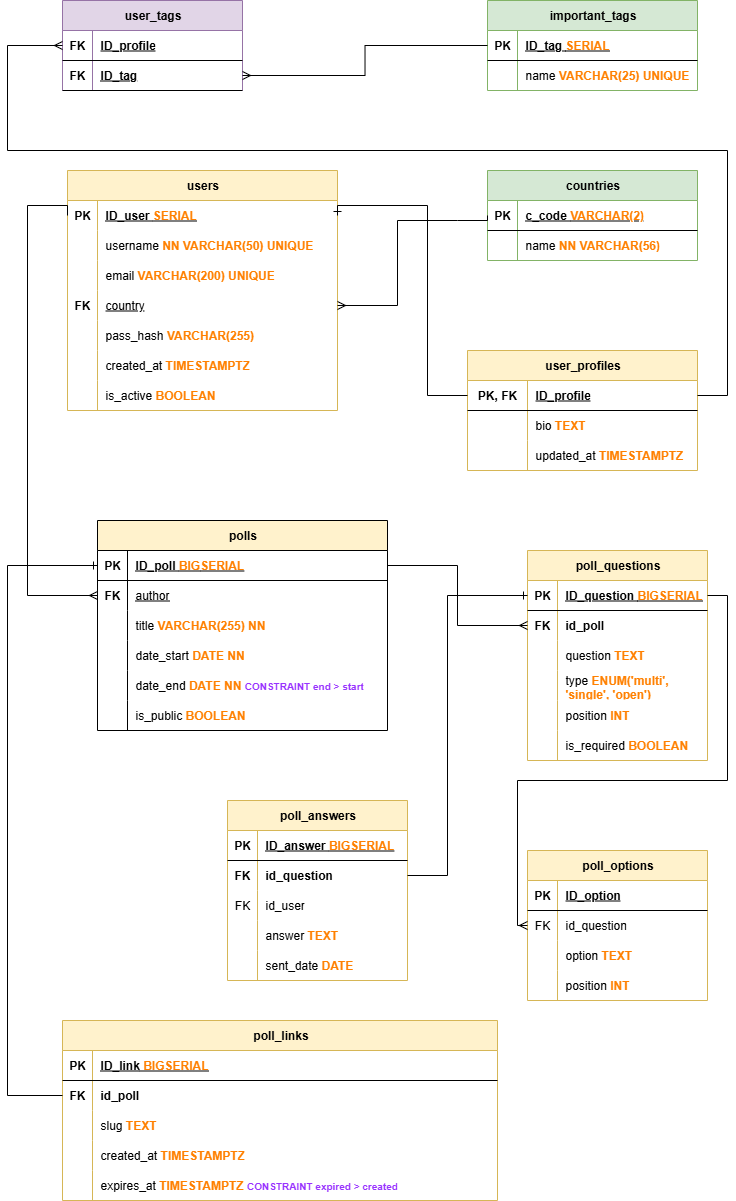
Each table is color-coded and, where applicable, includes a descriptive label in the header. For example, green tables represent dictionaries, while purple tables denote bridge tables. Main tables are highlighted in yellow.

Keys are listed on the left side of each table and are marked with abbreviations: PK for primary key and FK for foreign key. All key-related fields are defined as **NOT NULL**.

The database primarily employs **1:1**, **1:M**, and **M:N** relationships, which are visualized using connectors that link the corresponding keys between tables.

Each attribute is accompanied by a proposed PostgreSQL data type, although these suggestions are not binding at this stage. These types also imply general categories—for example, **VARCHAR** indicates a string (i.e., text).

## Logical Scheme



The poll-related data has been structured across five separate tables to decouple questions, their answer options, and the responses. A distinct table is also used for managing shareable links. This approach promotes data clarity, normalization, and long-term maintainability of the database schema.

## Objects

**users**

The Users table serves as a central element of the database schema, maintaining records of all individuals authorized to create polls or participate in voting. Each user is uniquely identified by a username and an email address. Passwords are securely stored using hashing algorithms.

The table tracks metadata such as the registration timestamp and whether the account is currently active. It also establishes a foreign key relationship with a countries lookup table, allowing users to specify their country of origin from a predefined list.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| Users | id\_user | This is an ID field that will auto increment, PK | Serial |
| username | This is an obligatory field and it cannot be null as this is relevant info which identifies the user. | Varchar with limitation to 50 chars. This length should be more than enough for the username. |
| email | Each user must have a unique e-mail address and it cannot be null. | Varchar limited to 200 chars. |
| country | This is an optional field to state the country of a user. It keeps a foreign key of a country dictionary table. Can be used for analytics later. FK from countries. | inherited |
| pass\_hash | This is the user’s password kept in a safe form. | Varchar as hash is basically a string, |
| created\_at | This is a timestamp of the time the user’s account was created. | TIMESTAMPTZ |
| is\_active | This record shows if the user’s account is active or has it been suspended or soft-deleted. | BOOLEAN |

There are basically two direct relationships that the users table is the part of.

1. The primary key from the users table (user\_id) appears in the polls table as a foreign key, identifying the author of each poll. This establishes a 1:M — one user can create many polls, but each poll has exactly one owner.
2. The users table links to user\_profiles in a strict 1:1 relationship: every user has exactly one profile, and each profile belongs to that same user.

Example with data

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id\_user | username | email | country | pass\_hash | created\_at | | is\_active |
| 1 | joedoe | doejo@gmail.com | DE | 5d41402abc4b2a76b9719d911017c592 | | 2017-01-08 04:05:06 | TRUE |

**user\_profiles**

Conceptually, user\_profiles is a child table of users. It stores records only for people who already exist in users, and every profile belongs exclusively to its corresponding user — hence the clear 1:1 mapping.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| user\_profiles | id\_profile | This is an ID field that will auto increment, PK | SERIAL |
| bio | This is an optional field a user can fill up with anything they want. | TEXT |
| updated\_at | This field stores the date of the last edit. | TIMESTAMPTZ |

The only relationship the user\_profiles table has got is the relationship with the table users.

Example with data

|  |  |  |
| --- | --- | --- |
| id\_profile | bio | updated\_at |
| 23 | I like it when it rains but it doesn’t rain often enough. | 2017-01-08 04:05:06 |

**countries**

This is a dictionary that lists all the existing countries that a user can choose as the place of their living.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| countries | c\_code | This is an ID field. It doesn’t auto-increment, because it stores a predefined string — the official code in alpha-2 for a given country. | VARCHAR(2) |
| name | The exact name for the code. Cannot be NULL. | VARCHAR limited to 56. |

As this is just a dictionary, its only relationship is that with the table users. The longest country name in the World is: *The United Kingdom of Great Britain and Northern Ireland*. This gives us 56 chars and that’s the reason for such a length in the data type. On the other hand, the c\_code field stores alpha-2 codes, which are always exactly two characters.

Example with data

|  |  |
| --- | --- |
| c\_code | name |
| DE | Germany |

**important\_tags**

To let users attach social-cause tags (or any other “important things”) to their profiles, we keep a separate **lookup table** that lists all allowed tags. Users pick from this list instead of typing free-form text, so the data stay clean and consistent.

The profile record itself doesn’t store tag names directly. Instead, you expose them through a view or a join—e.g., users ←→ user\_tags ←→ tags. That way the system always shows the up-to-date, canonical tag text while keeping the schema normalized.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| important\_tags | id\_tag | This is an ID field that will auto increment, PK | Serial |
| name | No name should be double of another. So, the field must be UNIQUE. | VARCHAR(25) |

Important\_tags is in an M:N relationship with user\_profiles, bridged by the junction table (e.g., user\_profile\_tags). One tag can appear in many profiles, and a single profile can hold many tags.

Example with data

|  |  |
| --- | --- |
| id\_tag | name |
| 5 | animal rights |

**user\_tags**

This is the junction table mentioned above for the relationship between the user\_profiles and important\_tags tables.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| user\_tags | id\_profile | FK from user\_profiles | inherited |
| id\_tag | FK from important\_tags | inherited |

Example with data

|  |  |
| --- | --- |
| id\_profile | id\_tag |
| 2 | 5 |

**polls**

This is the base table for all polls created by registered users. The basic information stored in the table includes the poll’s owner (the creator) and the poll’s title, which cannot be NULL, since the poll should be identifiable not only in the database by its ID but also in the web app. One can also track the start date of the poll, as well as the date when the poll should be locked. The last column indicates whether the poll is public (visible to all users) or private (available only to people with a link).

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| polls | id\_poll | This is an ID field that will auto increment, PK | SERIAL |
| author | identifies who created the poll; FK from the users table | inherited |
| title | human-readable name shown in the UI (must never be NULL; the ID alone isn’t friendly). | VARCHAR(255) |
| date\_start | timestamp marking when voting opens; not null | DATE |
| date\_end | timestamp after which the poll is locked. Not null. Should be constraint: the end shouldn’t be sooner than the start. | DATE |
| is\_public | TRUE = visible to everyone; FALSE = private, accessible only via a direct link. | BOOLEAN |

Poll has 1:M relationship with the table poll\_questions, 1:1 with poll\_links (one poll, one link) and 1:M with the table users (one user, many polls).

Example with data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id\_poll | author | title | date\_start | date\_end | is\_public | |
| 146 | 34 | Oranges or Potatoes? | 2017-01-08 | 2017-01-09 | | FALSE |

**poll\_questions**

This table stores every question created for any poll. Each question is linked to its parent poll by the poll\_id foreign key. Because polls can contain different question types (single-choice, multiple-choice, open-ended, etc.), the table includes a question\_type column that defines how each question behaves.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| poll\_questions | id\_question | this is an ID field that will auto increment, PK | SERIAL |
| id\_poll | FK from polls | inherited |
|  | question | question that a user should answer in the poll | TEXT |
|  | type | we can have a few types of questions in the poll: single-choice, multiple-choice or open-ended. Multiple will have more than one answer valid and therefore: will take more rows in the answers table later. | ENUM(‘multi’, single’, open’) |
|  | position | it indicated the sequence of questions in the poll (the position of the question in the poll) | INT |
|  | is\_required | is answering this question obligatory? | BOOLEAN |

There has been a Volunteers\_Events junction table implemented, because of the M:N relationship between those tables. Both of the connections are totally optional.

Example with data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id\_question | id\_poll | question | type | position | is required | |
| 56 | 22 | So, potatoes or oranges? | single | 1 | | TRUE |

**poll\_options**

When a user opens a poll question, they’re shown a list of answer options.  
Each option belongs to exactly one question, and each question belongs to exactly one poll.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| poll\_options | id\_option | This is an ID field that will auto increment, PK | Serial |
| id\_question | This is FK to poll\_questions. Identifies the question | inherited |
| option | This is one of the options from a given question to choose from | TEXT |
| position | This shows a sequence of options in a question | INT |

Consequently, the options table has a 1:M relationship with poll\_questions — one question can hold many options, but every option points back to a single question.

Example with data

|  |  |  |  |
| --- | --- | --- | --- |
| id\_option | id\_question | option | position |
| 2 | 56 | sometimes | 2 |

**poll\_answers**

This table records every response submitted by a user to a poll question. It’s the ledger that tells us who answered what and when.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| poll\_answers | id\_answer | This is an ID field that will auto increment, PK | SERIAL |
| id\_question | What question was a user answering? Foreign key to poll\_questions | inherited |
| id\_user | Who answered the question? FK to users table. | inherited |
| answer | Specific answer. Pretty straightforward when it comes to open-ended questions. For closed-choice questions, the answer stores the exact text of the option the user selected. In other words, the response is a verbatim copy of the chosen option’s label, making it readable even without joining back to the options table. | TEXT |
| sent\_date | When the question was answered and the poll sent. | DATE |

Example with data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id\_answer | id\_question | id\_user | answer | sent\_date |
| 12 | 63 | 15 | yes | 2017-01-09 | |

**poll\_links**

When a user decides to share a poll, the system creates a shareable URL that points straight to that poll.  
Inside that URL sits a slug — a short, human-readable token (e.g., best-pizza-vote). The slug uniquely identifies the poll in the link, making it easy to read, remember, and type while hiding the internal numeric ID. Bottom line: the slug turns a raw database reference into a friendly URL, and the table lets you manage those links cleanly.

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Field name | Field Description | Data Type |
| poll\_links | id\_link | This is an ID field that will auto increment, PK | SERIAL |
| id\_poll | FK linking to the polls table | inherited |
| slug | human-friendly URL | TEXT |
| created\_at | When was the link created? | TIMESTAMPTZ |
|  | expires\_at | When will it expire? Constraint: the expiration timestamp cannot be earlier than the creation time. | TIMESTAMPTZ |

Example with data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id\_link | id\_poll | slug | created\_at | expires\_at |
| 4 | 16 | happy-pizza | 2017-01-08 04:05:06 | 2017-01-09 04:05:06 | |