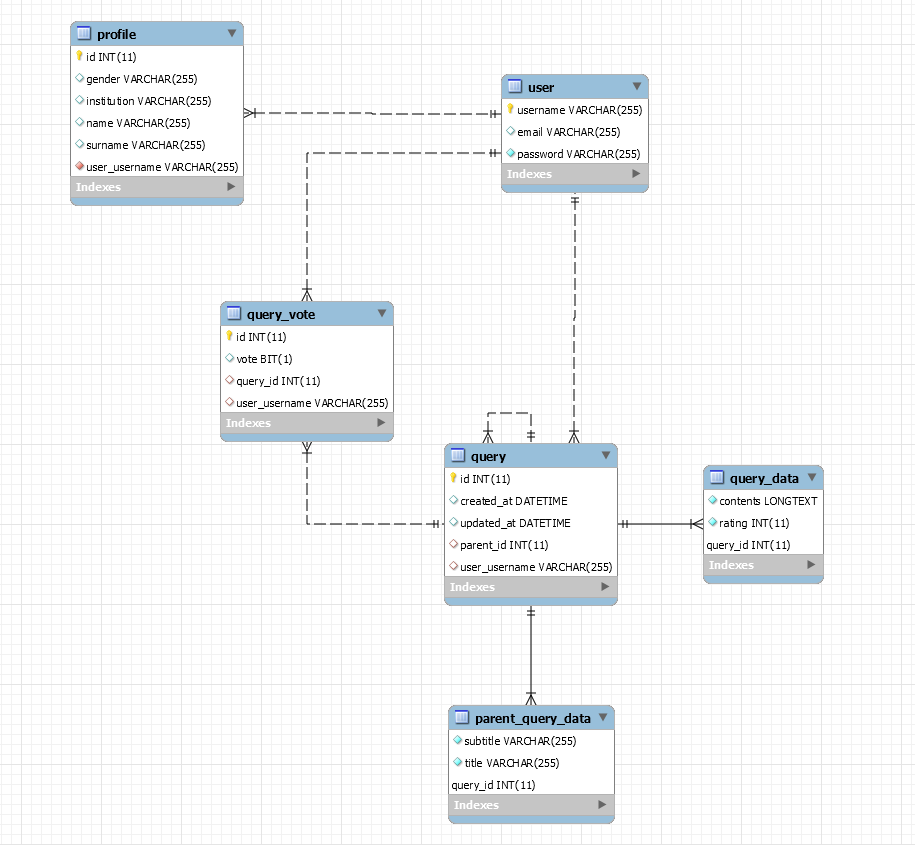
# ERD Overview

## ERD Diagram



## User-related functionality

There are two tables that are dedicated to the User family of classes at the moment, both quite simple. The **user** table stores a username, which is the primary key, an email and a password. The **profile** table stores an additional information about the user that is entirely optional. Note, that at the moment it has its primary key set as an integer; this will likely change to the username, so as to have profile ID mapped by user’s primary key. This is more of a stylistic change rather than a necessity, although it will reduce the size of the table by removing an, essentially, unnecessary column.

## Query-related functionality

A query itself is split into three tables – **query, query\_data** and **parent\_query\_data.** The reason behind this is a recursive relationship of query to itself. Query’s relationship to itself is denoted by a parent and children and together they form a tree. Every query can have children, and every query has a parent, except for the root of the tree, which cannot have a parent – an absence of parent is what defines it as the root of the tree. Parent has a set of unique, parent-specific attributes, specifically, a title and a subtitle. A title and subtitle were split into a separate table, and the reasoning behind this was as follows:

* Assuming that separate table was **not** created, a title and a subtitle could have been stored in the query itself and would have been nullable. However, this would add the overhead of storing two references that would be null for the majority of the cases except for a query is also a parent. This would add the size overhead not for the table only, but also for the Java class itself.
* Assuming that title and subtitle were stored in the **query\_data** table, the same issue remains – a table would contain two attributes who will potentially be null more often than not null, and it would just increase the size of the table and the Java class. Since evert single query entry has to have query contents stored in it, it would potentially add two unnecessary, always null values to every single query.

However, storing parent query data in a separate table adds an overhead of a single reference only to the Query Java class, and adds little to no overhead to the database itself, because if the query is not a parent, then no entry will be created in the database. Taking this and two previous points into consideration, it was concluded that introducing an additional degree of complexity to the ERD and class hierarchy is worth the space overhead that can be saved.

Apart from aforementioned design considerations, the query-related functionality is relatively straightforward – every query has a creation and update date, a **query\_data** entry that is mapped by query id and contains the contents of the query. A parent query would additionally have a **parent\_query\_data** entry, that is mapped by the query id and contains parent-specific information.

## Relationship between User-related tables and Query-related tables

Supporting tables for user (**profile)** have no direct relationship with any of the query-related tables, and vice versa. The user table and the query table have two separate kinds of relationship – a one-to-many relationship between the user and the queries, and a many-to-many relationship between the users and the queries.

A one-to-many relationship is a simple reflection of ownership – every query has to have a user that has created it, be it a parent or a child.

A many-to-many relationship is a reflection of the voting system – every user can upvote or a downvote any query, and the users’ vote are being tracked via this relationship; the vote itself is stored in a **query\_vote** table.

At the moment, **query\_vote** entries have its id generated sequentially. This will be changed as soon as possible, so as to have **query\_vote’s** primary key as a composite key consisting of user’s username and query’s id.