Database Plan

MegaMindz

Overview:

For our project we will be using PostgreSQL for our database in our web application.

Reasoning:

It was supported by all the tools that we already wanted to use.

We wanted to use python with Django, and PostgreSQL is one of the officially supported databases with djengo.

Databases

Django officially supports the following databases:

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

There are also a number of database backends provided by third parties.

This will be using the “psycopg2” library in python. Link to site: https://www.psycopg.org/

We also needed a database that is supported for Heroku since we need to use this tool to host our web server.

Heroku provides three managed data services to all customers:

- Heroku Postgres
- Heroku Redis
- Apache Kafka on Heroku

Things to Keep in Mind:

It will probably be best to coordinate with front end on data sizes. For example, if we have an email field, and we determine that we want to max at 100 characters, we should not allow the user to input more than 99 in the text box (one extra space for a null terminator character).

A function might also have to be made in order to escape single quotes (‘) out of string fields, since we need to pass single quoted strings to our database text fields. Note, this will be taken care of for you if you do string injections with the psycopg2 library (use %s in your sql command, and then passing it the parameter).

We are going to want to make these four functions for all of our database tables:

Add:

- Takes all of the fields (minus the ID auto generated field)
- Adds row to table with specified fields
- Returns ID (or -1 on fail)

Edit:

- Pass the ID for the row you want to change
- Pass all of the parameters (minus ID), those will be updated
    - Note, you will probably have to fetch these fields first, since you probably don’t know them
    - You could have specific edit functions that don’t require all of the fields, but that might result in a lot of functions. Maybe do this for a few frequently use one.
- Return success/fail4

Fetch:

- Pass ID
- Returns all of the rest of the fields

Delete:

- Pass ID
- Deletes the row
- Be careful

We will also need get functions that return a set of (or only one) row(s), based on a given set of fields. We can call these **“get”** functions

An example that would return one row: “get_user_info_with_login” that will take an email and password and return the user row that correlates to that login info (or -1 if doesn’t exist). (table: user_info)

An example that could return more than one row: “get_room_info_with_number” that will take a room number and return all the users in that room. (table: room_info)

For better performance, we can add an “INDEX” to the table that will specify tree structures to build the database in. For example, in a room_info table, we would probably want something like the following, so that our table is divided by room number first:

INDEX index1 (RoomNumber, ID, x_val, y_val)

You can also have more than one indexing if you have multiple often used queries. Then will be comma separated inside the “create table section”. Here is another one that could also go in that senerio.

INDEX index2 (x_val, y_val, ID)

In this situation you could give a range for x_val and y_val to find those near you. Though note, this will give you a square area, so you would have to do a little more on your end to cut out the edges into an actual circle.

Sample SQL File for Making User Info Table:

<see user_info.sql file in same folder as this pdf>

Note:

- table name words are separated with underscores like so: “table_name”.
- field names are camel case with first letter capital like so: “FieldName”