Midterm Design Challenge: Database

Author: Judrianne Mahigne

NJIT Email: jm2489@njit.edu

This is the DBMS description for the fall 2024 IT490 - Systems Integration midterm design project.

Table of Contents

Overview

This contains the information of the MySQL database. The database will store participant information, infection data, and location history. It will use a relational database as it is much more suitable for doing complex queries.

Database

For all intents and purposes, only one database will be refered to called `pandemicinfodb`. When dealing with sensitive information like health data, the company would need to be HIPPA compliant to avoid any legal issues. In this scenario, I will assume that all information related to the app is indeed sensitive and are being transported through secure channels.

Tables Schema

The table schema for the `pandemicinfodb` database is as follows:

Table: participants

```
+--------------+--------------+------+-----+---------+-------------------+
| Field        | Type         | Null | Key | Default | Extra             |
+--------------+--------------+------+-----+---------+-------------------+
| id           | int(100)     | NO   | PRI | NULL    | auto_increment    |
| age          | int(100)     | NO   |     | NULL    |                   |
| infected     | bigint(20)   | YES  |     | NULL    |                   |
| recovered    | boolean      | YES  |     | NULL    |                   |
| recover_time | bigint(20)   | YES  |     | NULL    |                   |
| TOD          | bigint(20)   | YES  |     | NULL    |                   |
+--------------+--------------+------+-----+---------+-------------------+
```

Table: locations

```
+---------------+---------------+------+-----+------------------+-------+
| Field         | Type          | Null | Key | Default          | Extra |
+---------------+---------------+------+-----+------------------+-------+
| id            | int(11)       | NO   | MUL | NULL             |       |
| latitude      | float(9,6)    | NO   |     | NULL             |       |
| longitude     | float(10,6)   | NO   |     | NULL             |       |
| created_at    | bigint(20)    | NO   |     | unix_timestamp() |       |
+---------------+---------------+------+-----+------------------+-------+
```

Table: infectionsByZip

```
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| zip            | varchar(10) | YES  |     | NULL    |       |
| infected       | bigint      | YES  |     | NULL    |       |
| last_reported  | bigint(20)  | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
```

Table: sessions

```
+---------------+--------------+------+-----+------------------+-----------
-----+
| Field         | Type         | Null | Key | Default          | Extra
|
+---------------+--------------+------+-----+------------------+-----------
-----+
| employee_ id  | int(11)      | NO   | PRI | NULL             |
|
| session_token | varchar(255) | NO   | UNI | NULL             |
|
| created_at    | bigint(20)   | YES  |     | unix_timestamp() |
|
+---------------+--------------+------+-----+------------------+-----------
-----+
```

Usage

To use the database, you can run queries using simple MySQL queries when making calls to the api. Of course, when handling the data, this is assuming that the requests to the api have been authenticated and authorized due to HIPPA regulations.

For example the following query using curl:

```
curl -X GET "https://www.pandemonium.com/api/listParticipants?
infected=not_null" \
     -H "Authorization: qwer1234apitoken \
     -H "Content-Type: application/json"
```

Will return the following response:

```
{"success":true,"data":[{"id":1, "age":33, "infected":173254324,
"recovered":true, "recover_time":174359834, "TOD":NULL}]}
```

The query will return data from the `participants` table where the `infected` column is `not null` in encoded JSON format. Any **CRUD** related statements work similarly.

The `sessions` table simply stores session tokens to validate api requests from the client. They are automatically updated for each employee when accessing information to the pandemicinfodb database. This also helps with any auditing and makes sure only certain resources are available on a "need to know" bases.

Running as a service

To run the script as a service, you can create a systemd service file in the `/etc/system d/system/` directory. Here's an example service file:

> dbServer.service

```
[Unit]
Description=PHP Database Socket Server
After=network.target mysql.service
Wants=mysql.service

[Service]
ExecStart=/usr/bin/php /path/to/your/dbServer.php
Restart=always
RestartSec=5
User=www-data
Group=www-data
StandardOutput=journal
StandardError=journal
WorkingDirectory=/path/to/your/directory/

[Install]
WantedBy=multi-user.target
```

The service file must be saved in the systemd service directory `/etc/systemd/system` for Ubuntu, RHEL, CentOS, and/or Fedora.

Then, you can enable and start the service with the following commands in the following order:

> The example provided is using Ubuntu 24

```
sudo systemctl daemon-reload
sudo systemctl enable dbServer.service
sudo systemctl start dbServer.service
```

You can also use `systemctl status dbServer.service` to check the status of the service.